68' MICRO JOURNAL

Australia A $ 4.75 | New Zealand NZ $ 6.50
Singapore S $ 9.45 | Hong Kong H $23.50
Malaysia M $ 9.45 | Sweden 30:-SEK

$2.95 USA

VOLUME VII ISSUE VII • Devoted to the 68XX User • July 1985

"Small Computers Doing Big Things"

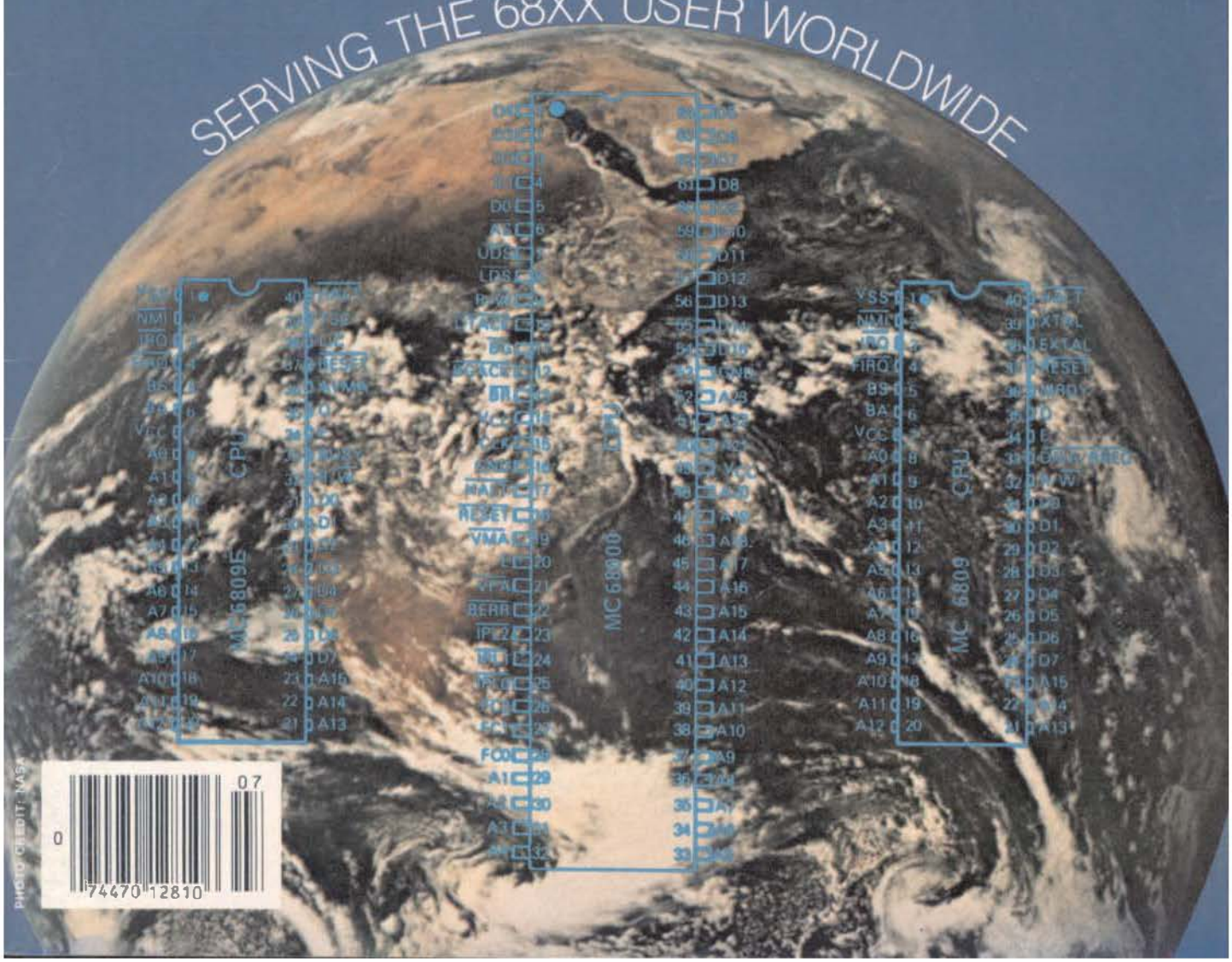


PHOTO CREDIT: NASA

0 74470 12810 07

WE DON'T PLAY GAMES

X-12+ A SERIOUS COMPUTER IN A DESKTOP PACKAGE

Multiprocessor Technology - Combination of 8, 16 and 32 bit types

1.0 Megabyte Memory - Insures no limitation on programs

"Winchester" Disk System - Fast response, large storage capacity

UniFlex* Operating System - The standard of comparison

Hardware Floating Point - Unmatched speed in a small system

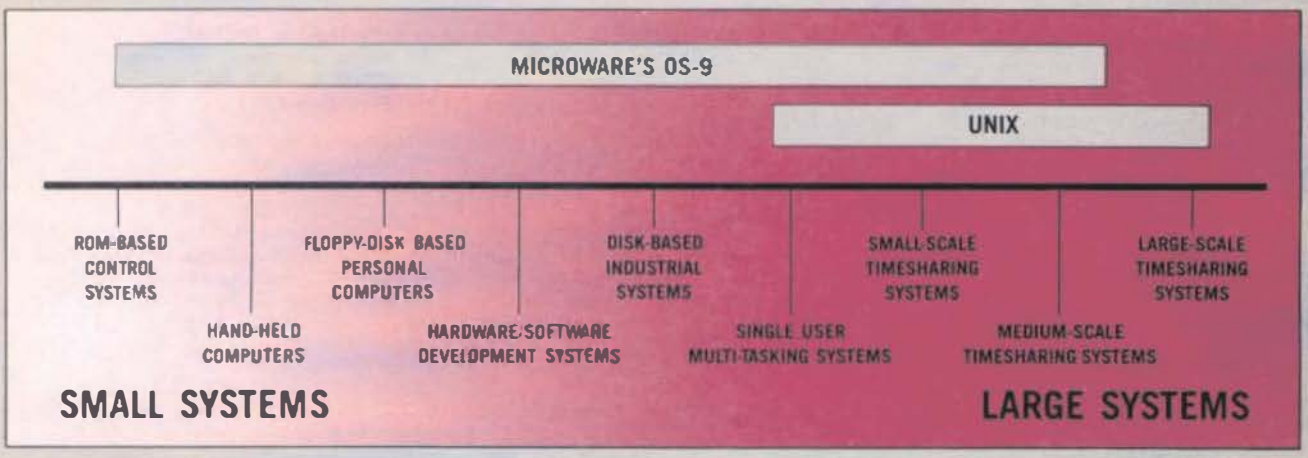
Up to Three Terminals - Instant expansion

*Trademark of Technical Systems Consultants

SWTPC

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. RHAPSODY
SAN ANTONIO, TEXAS 78216 (512) 344-0241

Only Microware's OS-9 Operating System Covers the Entire 68000 Spectrum



Is complicated software and expensive hardware keeping you back from Unix? Look into OS-9, the operating system from Microware that gives 68000 systems a Unix-style environment with much less overhead and complexity.

OS-9 is versatile, inexpensive, and delivers outstanding performance on any size system. The OS-9 executive is much smaller and far more efficient than Unix because it's written in fast, compact assembly language, making it ideal for critical real-time applications. OS-9 can run on a broad range of 8 to 32 bit systems based on the 68000 or 6809 family MPUs from ROM-based industrial controllers up to large multiuser systems.

OS-9'S OUTSTANDING C COMPILER IS YOUR BRIDGE TO UNIX

Microware's C compiler technology is another OS-9 advantage. The compiler produces extremely fast, compact, and ROMable code. You can easily develop and port system or application software back and forth to standard Unix systems. Cross-compiler versions for VAX and PDP-11 make coordinated Unix/OS-9 software development a pleasure.

SUPPORT FOR MODULAR SOFTWARE — AN OS-9 EXCLUSIVE

Comprehensive support for modular software puts OS-9 a generation ahead of other operating systems. It multiplies programmer productivity and memory efficiency. Application software can be built from individually testable software modules including standard "library" modules. The modular structure lets you customize and reconfigure OS-9 for specific hardware easily and quickly.

A SYSTEM WITH A PROVEN TRACK RECORD

Once an underground classic, OS-9 is now a solid hit. Since 1980 OS-9 has been ported to over a hundred 6809 and 68000 systems under license to some of the biggest names in the business. OS-9 has been imbedded in numerous consumer, industrial, and OEM products, and is supported by many independent software suppliers.

Key OS-9 Features At A Glance

- Compact (16K) ROMable executive written in assembly language
- User "shell" and complete utility set written in C
- C-source code level compatibility with Unix
- Full Multitasking/multiuser capabilities
- Modular design - extremely easy to adapt, modify, or expand
- Unix-type tree structured file system
- Rugged "crash-proof" file structure with record locking
- Works well with floppy disk or ROM-based systems
- Uses hardware or software memory management
- High performance C, Pascal, Basic and Cobol compilers

microware® OS-9™

MICROWARE SYSTEMS CORPORATION
1866 NW 114th Street
Des Moines, Iowa 50322
Phone 515-224-1929
Telex 910-520-2535

Microware Japan, Ltd
3-8-9 Baraki, Ichikawa City
Chiba 272-01, Japan
Phone (0473(28)4493
Telex 299-3122

'68' MICRO JOURNAL

Portions of the text for '68' Micro Journal were prepared using the following furnished Hard/Software:

COMPUTERS - HARDWARE

Southwest Technical Products
219 W. Rhapsody
San Antonio, TX 78216
S09 - 5/8 DMF Disk - CDS1 - 8212W - Sprint 3 Printer

GIMIX Inc.
1337 West 37th Place
Chicago, IL 60609
Super Mainframe - OS9 - FLEX - Assorted Hardware

EDITORS - WORD PROCESSORS

Technical Systems Consultants, Inc.
111 Providence Road
Chapel Hill, NC 27514
FLEX - Editor - Text Processor

Great Plains Computer Co., Inc.
PO Box 916
Idaho Falls, ID 83401
Stylograph - Mail Merge - Spell

CONTENTS

Send All Correspondence To:

Computer Publishing Center
68' Micro Journal
5900 Cassandra Smith Rd.
Hixson, Tn. 37343

Phone (615) 842-4600 or Telex 558 414 PVT 8TH

68' Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage Paid ISSN 0194-5025 at Hixson, Tn. and additional entries. Postmaster: send form 3597 to 68' Micro Journal, POB 849 Hixson, Tn. 37343.

Subscription Rates

1 Year $24.50 U.S.A., Canada & Mexico Add $9.50 a Year. Other Foreign Add $12 a Year for Surface, Airmail Add $48 a Year. Must be in U.S. currency.

Items or Articles For Publication

Articles submitted for publication should include authors name, address, telephone number and date. Articles should be on either 5 or 8 inch disk in STYLOGRAPH or TSC Editor format with 3.5 inch Column width. All disks will be returned. Articles submitted on paper should be 4.5 inches in width (including Source Listings) for proper reductions. Please Use A Dark Ribbon!! No Blue Ink!! Single space on 8X11 bond or better grade paper. No hand written articles accepted. Disks should be in FLEX2 6800 or FLEX9 6809 any version or OS-9 any version.

The following TSC Text Processor commands ONLY should be used: .sp space, .pp paragraph, .fi fill and .nf no fill. Also please do not format within the text with multiple spaces. We will enter the rest at time of editing.

All STYLOGRAPH commands are acceptable except .pg page command. We print edited text files in continous text form.

Letters To The Editor

All letters to the editor should comply with the above requirements and must be signed. Letters of "gripes" as well as "praise" are solicited. We reserve the right to reject any submission for lack of "good taste" and we reserve the right to define "good taste".

Advertising Rates

Commercial advertisers please contact 68' Micro Journal advertising department for current rate sheet and requirements.

Classified Advertising

All classified ads must be non-commercial. Minimum of $9.50 for first 20 words and .45 per word after 20. All classifieds must be paid in advance. No classified ads accepted over the phone.

GIMIX HAS THE 6809 SYSTEM TO SUIT YOUR NEEDS

HARDWARE

All systems feature the **GIMIX CLASSY CHASSIS**; with a ferro-resonant constant voltage power supply, gold plated bus connectors, and plenty of capacity for future expansion.

Static **RAM** and double-density **DMA** floppy disk controllers are used exclusively in all systems.

All systems are guaranteed for 2 MHz operation and include complete hardware and software documentation, necessary cables, filler plates, etc.

Systems are assembled using burned-in and tested boards, and all disk drives are tested and aligned by **GIMIX**.

You can add additional components to any system when ordering, or expand it in the future by adding **RAM**, **I/O**, etc.

GIMIX lets you choose from a wide variety of options to customize your system to your needs.

SOFTWARE

All **OS-9/FLEX** systems allow you to software select either operating system.

Also included is the **GMXBUG** monitor and, in systems with 128K or more of RAM, **GMX-VDISK** for **FLEX**.

All **GIMIX OS-9** systems include **Microware's Editor, Assembler, Debugger, Basic09,** and **Runb**; and the GMX versions of **RMS** and **DO** for **OS-9**.

All **GIMIX** versions of **OS-9** can read and write RS color computer format OS-9 disks, as well as the Microware/GIMIX standard format.

New and exclusive with **OS-9 GMX III** systems is the **GMX OS-9 Support ROM**, a monitor for **OS-9** that includes memory diagnostics and allows the system to boot directly from either hard disk or floppy.

A wide variety of languages and other software is available for use with either **OS-9** or **FLEX**.

OS-9 GMX III/FLEX SYSTEMS (#79)

The **#79** super system now includes (in addition to the above): the **GMX 6809 CPU III**, a **256K CMOS Static RAM Board (#72)**, and a **3-port Intelligent Serial I/O Processor (#11)**.

The **GMX 6809 CPU III** can perform high-speed DMA transfers from memory to memory and uses memory attributes and illegal instruction trapping to protect the system and users from program crashes. If a user program crashes, only that user is affected; other users are unaware of the problem.

The 3-Port Intelligent Serial I/O Board (#11) significantly reduces system overhead by handling routine I/O functions; freeing the host CPU for running user programs. This improves overall system performance and allows user terminals to be run at up to 19.2K baud.

with dual 40 track DSDD drives	$5998.79
with dual 80 track DSDD drives	$6198.79
with #88 dual 8" DSDD drive system	$7698.79
with #90 19MB Winchester subsystem and one 80 track	$8898.79
with a 47MB Winchester subsystem and one 80 track	$10,898.79
with a 47MB plus a 6MB removable pack Winchester subsystem and one 80 track drive	$12,398.79

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if order is under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 6 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account #73-32033.

OS-9 GMX I / FLEX SYSTEMS #49

The **#49** systems include 64KB static RAM, #05 CPU, #43 2 port serial board.

with dual 40 track DSDD drives	$3998.49
with dual 80 track DSDD drives	$4198.49
with #88 dual 8" DSDD drive system	$5698.49
with #90 19MB Winchester subsystem and one 80 track	$6898.49

OS-9 GMX II / FLEX SYSTEMS #39

The **#39** systems include 128KB static RAM, #05 CPU, #43 2 port serial board.

with dual 40 track DSDD drives	$4498.39
with dual 80 track DSDD drives	$4698.39
with #88 dual 8" DSDD drive system	$6198.39
with #90 19MB Winchester subsystem and one 80 track	$7398.39

GIMIX DOES NOT GUARANTEE PERFORMANCE OF ANY GIMIX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

EXPORT MODELS: ADD $30 FOR 50Hz. POWER SUPPLIES.

GIMIX, Inc. reserves the right to change pricing, terms, and products specifications at any time without further notice.

ALL PRICES ARE F.O.B. CHICAGO

Contact GIMIX for price and availability of UniFLEX and UniFLEX GMXIII Systems.

NOTE on all drive systems: Dual 40 track drives have about 700KB of formatted capacity; dual 80's about 1,400KB; dual 8" about 2,000KB. The formatted capacity of hard disks is about 80% of the total capacity.

Want to expand your system to a megabyte of Static RAM and 15 users?

Simply add additional memory and I/O boards. Your GIMIX system can grow with your needs. Contact us for a complete list of available boards and options.

#72 256KB CMOS STATIC RAM board with battery back up	$1898.72
#64 64KB CMOS STATIC RAM board with battery back up	$528.64
#67 64KB STATIC RAM board	$478.67
#11 3 port intelligent serial I/O board	$498.11
#43 2 port serial I/O board	$126.43
#42 2 port parallel I/O board	$88.42
#95 cable sets (1 needed per port), specify board	$24.95

NOW SHIPPING !

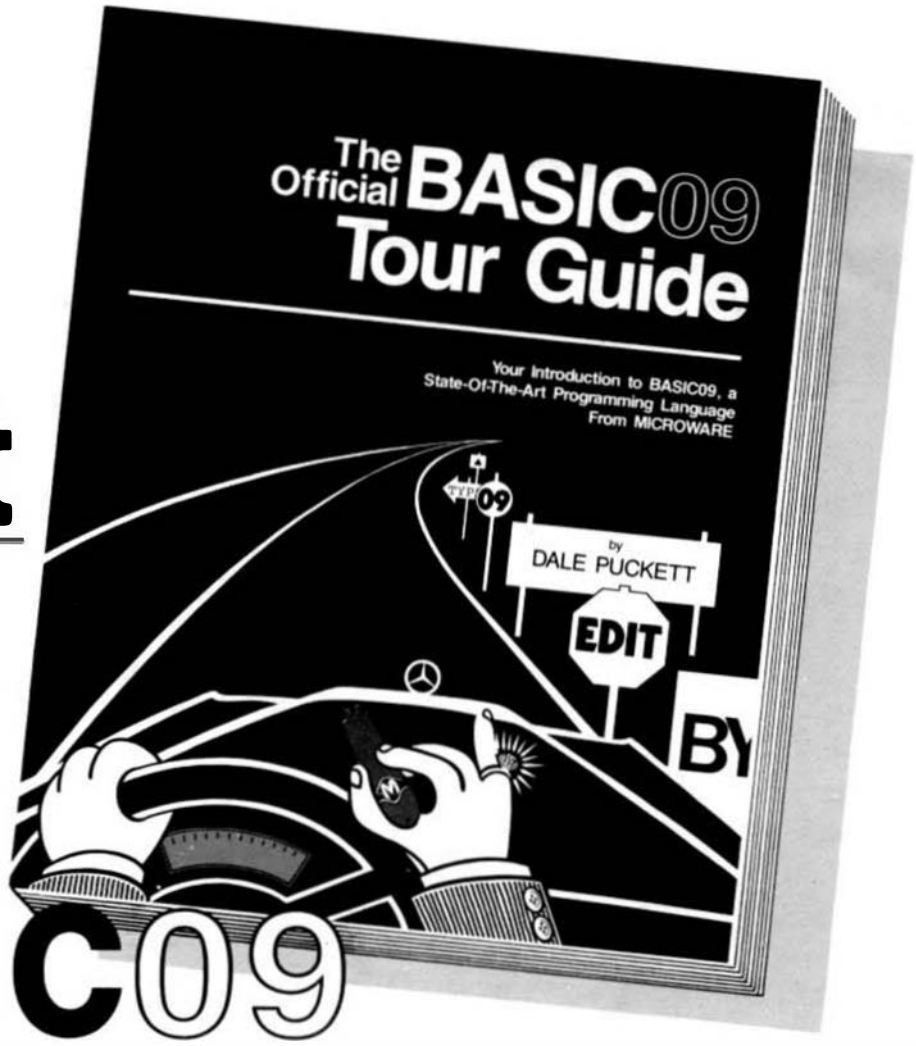
UniFLEX

GMX III Systems

GIMIX INC.

1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055

Get the most out of BASIC09



The **OFFICIAL BASIC09 TOUR GUIDE** is skillfully written in a friendly and easy-to-read style. Just perfect for those new to computers and to BASIC09. It's also a *valuable reference book* for programmers, engineers, students and hobbyists, providing an in-depth look at BASIC09 plus an overview of the OS-9 operating system. Comprehensive reference sections on BASIC09 and OS-9 commands are also included.

The book "maps" out your route through the Mercedes of Basics . . . BASIC09 and puts you in the driver's seat in no time. Fasten your seatbelt, sit back and enjoy the ride to perfecting your programming skills.

MICROWARE . . .

The **OFFICIAL BASIC09 TOUR GUIDE** comes from the people who wrote BASIC09. As the leader in 6809 system software, we at MICROWARE care about our users and want to help you get the most from our products.

It's Easy to Order.

Phone orders are accepted from MasterCard or VISA cardholders or for COD shipment. You can also order by mail using the coupon below. Quantity discounts are available to educational organizations and dealers. For further information contact Microware.

Microware Systems Corporation
1866 N.W. 114th Street
Des Moines, Iowa 50322
Telephone 515/224-1929
Telex 910-520-2535

Please send _____ copies of the **Basic09 Tour Guide** book at $18.95 each. Add $2.00 for UPS shipping in the U.S. or $5.00 for overseas air mail per book. Iowa residents add 4% sales tax.

Name _____

Address _____

City _____

State _____ Zip _____

☐ I have enclosed a check

☐ Charge to my bank card:
MasterCard ☐ VISA ☐
Card Number _____
Expiration _____

microware®

Specialists in system software for 68-family microprocessors since 1977.

OS-9 and BASIC09 are trademarks of Microware and Motorola

..HEAR YE.....HEAR

OS-9™ User Notes

By: Peter Dibble
As Published in 68 Micro Journal

The publishers of 68 Micro Journal are proud to announce the publication of Peter Dibbles **OS9 USER NOTES**.

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9

HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootsrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9

DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages

Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include

No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled **Operating Programs.** The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing **your OWN** more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

!!! Coming Soon !!!
Catch Us Next Month for More Details

Continually Updated In 68 Micro Journal Monthly

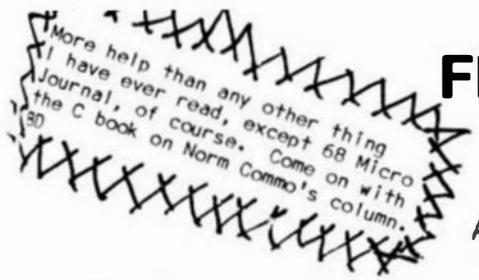
VISA

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN. 37343

(615) 842-4600
Telex 558 414 PVI

TM - OS9 is a trademark of Microware Systems Corp. and Motorola Inc.
TM - 68' Micro Journal is a trademark of Computer Publishing Inc.

HEAR YE.....HEAR YE



FLEX™ USER NOTES
THE 6800-6809 BOOK

By: Ronald W. Anderson
As published in 68 MICRO JOURNAL™

The publishers of 68 MICRO JOURNAL are proud to announce the publication of Ron Anderson's **FLEX USER NOTES,** in book form. This popular monthly column has been a regular feature in 68 MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. Now all his columns are being published, in whole, as the most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

As a SPECIAL BONUS all the source listing in the book will be available on disk for the low price of: FLEX™ format only — 5" $12.95 — 8" $16.95 plus $2.50 shipping and handling, if ordered with the book. If ordered separately the price of the disks will be: 5" $17.95 — 8" $19.95 plus $2.50 shipping and handling.

Listed below are a few of the **TEXT** files included in the book and on diskette.

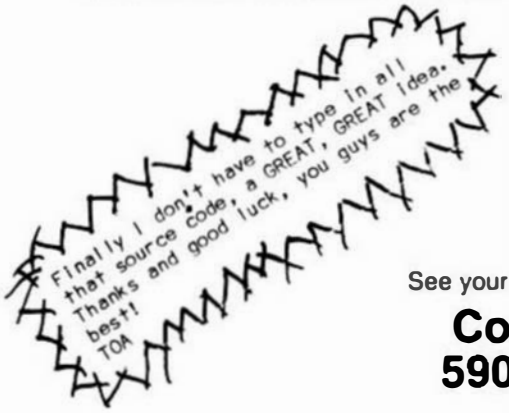
All TEXT files in the book are on the disks.

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC
	(And many more)

**Over 30 TEXT files included in ASM (assembler) — PASCAL — PIC (position independent code) TSC BASIC-C, etc.

NOTE: .C1, .C2, etc. = Chapter 1, Chapter 2, etc.

This will be a limited run and we cannot guarantee that supplies will last long. Order now for early delivery.



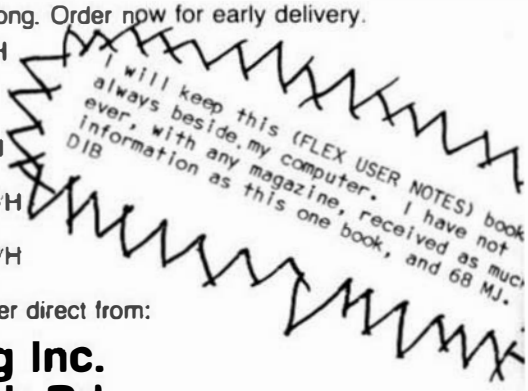
Foreign Orders Add $4.50 S/H

Softcover — Large Format

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H



See your local S50 dealer/bookstore or order direct from:

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343
(615) 842-4601

TELEX 558 414 PVT BTH

**FLEX is a trademark of Technical Systems Consultants

FLEX User Notes

Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, Mi 48105

Twenty Years

I guess twenty-two would be more accurate. I was looking at the "Life Science Library" the other day (copyright 1963), and I decided to see what they had to say about computers. There is a photo titled "Magic Crystals and a Trend to Miniatures". The statement is made "Other crystals being developed are so diminutive that one of them, no larger than this capital 'O', does the job of four transistors and two resistors." Of course the capital "O" in question is big enough in terms of present day technology to hold about a quarter of a 2732A ROM chip, which means that it would hold about 8000 memory locations and therefore at the very least twice that many transistors and associated other components. We've come a long way!

What is FLEX

Based on input from Don Williams, I am going to start devoting some space here to some information for newcomers to FLEX. I thought it might be a good idea to start at the beginning, though I will probably move rapidly. What is a Disk Operating System (DOS) anyway? As you all most likely know, a microprocessor can do absolutely nothing unless it is running a program. If you have a Color Computer, you probably know that it runs BASIC when it is powered up. The BASIC for the Color Computer includes a disk operating system. If you have an SS-50 bus computer, you probably realize that it comes up running a "monitor" program in ROM. In both cases, the computer starts up in the "command" mode. That is, it is waiting for instructions from the user. The monitor (in the case of the SS-50) contains a few rudimentary functions that allow you to look at the contents of memory or run a quick memory test, but in general it can't do very much other than wait for a command from you (of course it has to communicate with the terminal to do that).

Now, when you type U or D on that SS-50 (or most of the single board computers), you are telling the system to "boot" the DOS from the system disk in one of your disk drives. The monitor contains a very short program that tells the disk controller to read sector 0 on track 0 into some fixed location in memory, and then to jump to the first location of the code that it read in. The boot program on the disk is larger, and it then instructs the disk controller to read in all of FLEX and jump to its COLDStart address. In other words, monitor has a little boot program (boot is an abbreviation for bootstrap, as in "lifting yourself up by your own bootstraps"), it loads a considerably bigger boot program from the disk, and that loads the DOS and starts running it.

Why do you need a DOS? You might guess by the name "Disk Operating System" that it has something to do with the floppy or hard disk drives in the system. Having a disk drive would do you no good without means to read programs and disk files from the disk and write files to the disk. The DOS also provides facilities to prepare a disk to accept program and data files from the computer. It allows you to copy the contents of a disk to another disk. It structures the disk so that it has a "directory" so that you can keep track of the files on the disk. One of the early 6800 operating systems did not have that luxury. The user had to tell the system to read the file at track 07 and sector 04. He had to keep track of what was on the disk with a pencil and paper. Perhaps in the days of single sided single density 5" disks, that was almost practical. After all there were only 340 sectors on the disk, about 85K of storage. Now with double sided double density 40 track disks

with 1400+ sectors, or the 80 track versions with 2800 sectors, that directory method would not be practical at all.

FLEX includes a large number of "utility commands" that reside on the system disk and are supplied with it. The distinction is made between "memory resident" commands and "transient" commands. There are only two memory resident commands in FLEX. they are called GET, and MON. The first will load a "binary" file to memory and the second exits FLEX and returns to the system monitor program. All other utilities (at least as FLEX is supplied) are "loaded" from the disk and then "executed" (fancy word for "run"). Flex has provision for re-routing of input and output, so that information may be input from the terminal or a disk file, and output may be to the terminal, a printer, or a disk file. You may extend that selection of input/output devices by means of software, to include such accessories as a modem, multiple printers, etc.

What are the advantages of FLEX over some other disk operating systems? That is hard to answer. What might be a definite advantage to me might be a disadvantage to someone else. Basically, though, here is why I like FLEX. It is SIMPLE and SMALL. It doesn't create complexities or roadblocks to my accessing input and output devices directly with programs that I write. It is relatively "transparent". I can easily write a program that will run on my computer that is 100% independent of FLEX, and then dump that program into a ROM and run it in "stand alone" hardware, in other words, a special purpose computer.

The disadvantages, depending on your viewpoint might be just what I think are the advantages. Some of the larger and more advanced operating systems (OS-9 and UNIFLEX) are much larger and more capable. FLEX has just one directory. The others allow you to have tree structured directories. In other words, you can have sub-directories and sub-sub-directories several levels deep. That is a great convenience when you have a hard disk that can hold literally a couple thousand files. The more advanced operating systems usually have features called multi-tasking and multi-user. Multi-tasking means that you can have the system compiling or assembling a program as a "background" task while you are editing a file. You can print a program listing or a letter (or a book) while you are using the computer to do something else. Of course the computer has only one processor, and it can do only one thing at a time, but the system "slices" time up into chunks and allots a slice to each of the tasks that are running. Multi-user means that the tasks include those necessary to support more than one user on the system.

FLEX supports a very simple multi-task. It has a feature called "print spooling". You can write output that you want eventually to go to your printer to a temporary disk file, and then print it from that file while you are doing something else. Generally the operation of print spooling is unsatisfactory at least to me. If I try to edit a file while print spooling is occurring, I find that the computer misses my input characters frequently because it is off doing something else while I am typing. I therefore don't use print spooling. I've heard from others who don't type as fast as I, who have no trouble using that feature, however.

Perhaps that is enough of an introduction for one time. I'll continue next time with some detail about what is in FLEX and how to use it effectively.

Feedback

Last month's column contained a plea for some reader feedback. The FLEX information above is given in response to words from Don Williams, as I said above. I have in front of me another letter that is quite in the opposite direction. I won't quote the whole thing, but essentially it says...

OK so now I have one of these SS-50 bus systems with 56K of memory, FLEX, and "a ton of software for it", a CRT, a Printer, A modem, and several special cards hooked up to the 30 pin buss. Now what do I do?

The letter says that the writer would like to go on to add a hard disk and a lot more RAM. He would most of all like some "multi-programming" capability. He goes on to say that FLEX is great for a single user system, but it would seem to stand in the way of larger memory and multi-user and multi-tasking operation.

My personal advice won't satisfy this reader, since I would say why do you need

multi tasking on a personal computer? I would tell someone with a nice system such as the one described in the letter to go on and use the heck out of it. Are you an accomplished programmer in assembler, Pascal, Fortran, PL/? and "C"? If not, there are lots of things you can do with the system at this point.

Of course FLEX stands in the way of multi-user and multi-tasking operation. The beauty of FLEX is, as I have said before, that it is SIMPLE. My simplistic answer to multi tasking is to put another computer alongside of my primary one. I have a PT69 system with a pair of DSDD 40 track drives and my old ADM-3A sitting next to my "big system" that has a pair of 8" DSDD Qume drives on it. If I have need to compile a program and write a letter at the same time, I just start one of them compiling and go use the other for editing. I have a serial link between them and with TX and RX utilities published a couple months ago, I can easily transfer a letter or a program back and forth between systems freely.

None of my "raving" in the last paragraph is really an answer to the basic question brought up in the letter. There are several large capacity memory boards on the market. The one with which I am familiar is the one from Computer Excellence. I have their 256K version installed in one system at work and it is great as a virtual disk drive. They publish information in their manual for using the memory in a multi tasking mode as well.

This magazine had ads for some time for a product called Dynashare which would allow multi-users on a system with extended memory running FLEX. I don't know what happened to that product (Don insert a note if you can shed any light on this). ***(see footnote-DMW)

Certainly there are people out there capable of writing a few articles on the realities and practicalities of UNIFLEX. ***-(answer here: Simple, never could get anyone to do a UniFLEX column) You have a column right here on OS-9 ***-(answer here: Microware gave effort and support towards the OS-9 column, the difference). As the writer of the letter pointed out, of course, this means that you start over on software, something I at least, am not willing to do. ***-KBASIC allows software porting FLEX to OS-9 - DMW!

What I see happening with the 6809 is that there will be little or no more software written for our systems. That doesn't bother me, I still have plenty that I can use to do what I want to do with my system. However, this should certainly be a consideration when it comes to investing more in one of these systems. What am I going to do with my systems? The PT69 with the drives, the terminal and the Epson printer will go away to College with my daughter next Fall. If someone actually does come up with a hard disk (simple plug in the controller and run it version) and some drivers that can be added or appended to FLEX for it, I'll put one on my "large system". Where that PT69 now sits, I wouldn't be too surprised to find a Mac or an IBM clone (or one of each) one of these days.

I've written my own text processor, designed to be just what I need and no more or less. I'm working on my own screen editor (reason enough to keep this system busy for a couple of years more), and I use this system to develop 6809 software for stand-alone systems as I've said before.

I don't mean to favor one supplier over the others, but have you seen the latest Peripheral Technology offering? A complete system with a hard disk for $2000! What a nice development system, word processor, or what have you.

Now, with some of you wanting me to go back to the beginning and describe FLEX, and some of you saying you are tired of your system and where do you go from here, WHERE DO I GO FROM HERE????

Actually it is more complex than I've indicated so far. Out there somewhere is a group who call themselves something like the "Society for the Promotion and Encouragement of Miniflex Users in America" (with apologies to the S.P.E.B.S.Q.S.A.).* I received a letter recently from someone who complained that all the software I talk about is for the 6809, and he is still running FLEX2. He went on (mistakenly) to say that most of the programs submitted to '68' are 6800 code. Actually, the major contributors of utilities, Leo Taylor, Bruno Puglia and a few others, are kind enough to write their utilities in 6800 code

structured so that by changing a constant or two, they may be assembled as FLEX2 utilities or as FLEX9 utilities. That practice doesn't indicate that there are more 6800 users than 6809 users reading '68' Micro Journal.

Next in the spectrum are the 6809 beginners, possibly recently increased in numbers by the demise ***(see footnote #2-DMW) of the Color Micro Journal and the switch of subscriptions to this journal. I think I come in the next category, that of advanced 6809 user. I am, however rather content with the state of advancement of my system(s). Last are those who now have a "full" 6809 system running, and have tried all the available software, and are "bored" and want to move on to something bigger. I cautiously say that SOME of the last category are collectors of equipment and/or software who, now that there is little new software appearing for the 6809 systems, want to move on to bigger and better things to collect.

Obviously I can't please the entire spectrum, at least not every month. Please consider my request last month for input, and the continuation of that appeal this month as a survey. Tell me what YOU want, and I'll try to proportion the discussions so that the most wanted items get covered most, and the lesser asked for topics don't get left out altogether.

As of this writing, the April '68' has been out for a week or two, and two readers have responded to my discussion of sorting. Believe it or not, it was pure chance that led Ron Voigts in his "BASIC OS-9" column and I to discuss sorting in the same issue. There must be several versions of each type of sort, as Ron's descriptions of the plain bubble sort, the insertion sort, and the Quicksort algorithms vary somewhat with my previous understanding of them. Of course there are many variations of each type of sort. One variation of the insertion sort that I have seen takes the items in their original order and inserts them in an array in order. That is, first item goes in at the top. Second item goes either above or below the first, depending on whether it is bigger or smaller. Third item goes top, bottom, or between the first two. Of course that means a lot of "moving down" of the items below the one being inserted.

There is a nice trick variation of this kind of insertion sort. You can use a binary search for the place to insert the new item in the so-far sorted list. Suppose you have already sorted 1000 items and you have item 1001 to insert. Test at item 500. If the new item is smaller than item 500 test at 250. If it is larger, test at 750, etc. each time halving the interval in which the final insertion will be made. With 1000 items, you can find the insertion point in ten compares. Much better than running down the whole list one item at a time until you find the place. You can, if you stop to think about it, perform this sort in one array. Top item becomes the first and only item in the sorted list. Next item moves from unsorted list to sorted by remaining where it is or being moved above the first item, etc. The number of items doesn't change, just the boundary between the sorted list and the unsorted one. It turns out that the insertion sort with binary search is fairly complex to implement. It runs about 3 or 3.5 times faster than the bubble sort I published in April. The sort I called "Half Shell" runs about 5 to 7 times faster than the insertion sort with binary search, and it is easier to implement.

I am going to hold off further sort program listings until I see if there are more than two responses to that question.

Disk Compatibility

I've been receiving a large amount of feedback from readers and software suppliers regarding the FLEX disk compatibility problems. There is not enough space left in this column for all of it, and the most complete analysis of the problem arrived today. I'll include an update in the next column. Meanwhile, I ought at least to say that the patch reported a few columns ago, (from Kent Meyers) will only work with a 1771, 1773 type controller. The newer ones, including the 2797 used in the Peripheral Technology system, can't be made to ignore the side byte, and can't be patched by hardware. Meanwhile the old "standard" 340 sector SSSD 35 track for 5" disks, and the SSSD 15 sector per track, 77 track format for 8" disks are compatible with everyone's FLEX and hardware.

Perhaps the best way would be for someone to write a set of format utilities that are compatible with all the

controllers, and then to publish the hardware patch to the SWTPc controller boards to make the side select the side select and the density tbe density per the IBM Standard. Then we could all throw away the SWTPc type of Newdisk utility, and be compatible all the way. I understand that Peripheral Technology is changing their board so that it may be jumper (or switch) programmed to be compatible with either format. (That is only rumor, don't hold me to it). Apparently while I've been wondering about the compatibility problem, a lot of folks have known exactly what the problem is. I'm glad I brought the subject up. By the time the next column is in preparation, I ought to be able to get permission to quote one or two of the responses I've received.

- - -

***-**Footnote #1:* It appears that we were the only ones selling or pushing Dynashare, some time back. It had some 'nasty' bugs that showed up when a hard disk or other device, in the same address range was installed on the systems. Also it hard-set the end of memory pointer to $BFFF, which caused a lot of headaches if you had printer drivers, etc., residing below in memory.

The folks who actually did Dynashare for Computer Systems Center of Chesterfield. Mo., were traveling, or something (per a few calls from the author), and fixes were slow and beyond what I felt was sufficient for a commercial product. So, I dropped it from the catalog. However, since then I have received a new version from Joe Turner, which is supposed to have the bugs fixed. I just have not had time to sit down and debug the product a second time. S.E. Media now has the 'new, improved' (fixed) version in stock, but I **will not** advertise it until I know it is running properly!

Actually it is a shame, for I felt that Dynashare should have been a '-mark' piece of software, it had all the promise of making FLEX even better. While it was working, it did, when it died, so did a lot of files that were open! I am willing to send a 'beta copy' of the latest version to someone willing to give it a real thrashing. That means you must have an SWTPC with at least 128K (SO-9 SWTPC system or better) and a SWTPC hard-disk. I insist on the hard-disk as most anyone serious enough to want multi-user multi-tasking operation, must have need for the hard-disk. Let me know if you are interested and what qualifications you have for 'proving' a software project.

So Ron, to answer your question above; I am keeping quiet and not allowing S.E. Media to advertise it (despite having bought and stocking it) until **I AM SATISFIED IT IS WORKING PROPERLY - AND AS ADVERTISED!** Betcha heard that before.

- - -

***-**Footnote #2:* I take a little exception to the word 'demise'. COLOR Micro Journal did not 'demise'. I stopped it because it was going nowhere.

Fact is, it was in the black and a profit maker (not much but plus) when I put it in the 'HALT Mode', and that is where it is now. We think we were the 2d largest color computer magazine out there, when all the figures were compiled. And growing each month. But, I could not see it going anywhere, and I needed the people and facilities it was using, to do other things. You see, I have very little faith in Tandy to continue the color computer, as we now know it. I am afraid the next generation CoCo will be too high priced for the average CoCo user, as we know him/her now days. **But, if not and it goes well, COLOR micro Journal will be back.**

We still have the loyal readers, who were interested in the serious aspects of the thing (not games or toy users). I can have it back rolling on 45 days notice! But, I am going to have to have not only the readers, but also advertisers, which we were short on before. It was just that we catered to a market that could not advertise. And many of those that were, or did, went belly-up, and guess who sometimes ended up holding the empty bag? **The very sad part of it is that many of those that did not make it, had the better products!!!** They had the stuff that made the CoCo a 'real computer'. But I guess that is the problem, no one, including Tandy, wants the CoCo to be a 'real computer', that is not in it's present form. Hence, we and a very few others were out there, all alone. Sorta like 'Maytag' repairmen - a lonesome situation. **BUT, I SAY AGAIN, WE CAN RETURN!**

Also See Editor's Comment on page 45

DMW

OS-9 User Notes

Peter Dibble
517 Coler House
Rochester, N.Y. 14620

It's Only Virtual

Virtual memory has begun to be an issue in our little corner of the world. TSC has a version of Uniflex for the 68010/20 that supports virtual memory. Microware is working on a similar version of OS-9/68K. I like virtual memory. It is a wonderful labor-saving device. There is, however, a dark side...

Virtual memory is a new concept for microcomputers. Let me start with a review of the "virtual" business.

A virtual resource is something that appears to a program to exist, but is actually an imitation constructed by the operating system and (perhaps) some hardware tricks. In the case of virtual memory, a program can read and write memory that doesn't actually exist.

Virtual memory requires two special features to be implemented in hardware. The simpler one is something like the DAT found on level two systems except that it works in reverse. Say we are using a processor with a 24-bit address buss on a system with one megabyte of real memory. A megabyte of storage is only a 20-bit address range. The DAT must translate each 24-bit address produced by the processor into a corresponding 20-bit address or generate an error, a "page fault."

The other special hardware required for virtual memory is a special way of handing page fault interrupts. A page fault might occur when an instruction was being fetched, or it might happen while operands were being read. Sometimes an instruction might be well under way before a page fault takes place. The problem is that the instruction might already have had some effect. An instruction that modifies a string many bytes long might hit a page fault after operating on most of the string. You can't always just restart the instruction from the beginning when you are ready. The usual way of solving the problem is for the processor to save its low-level state as well as the usual registers when it gets a page fault. A special return-from-interrupt instruction restores the state and restarts the instruction where it left off.

Systems, like the 68010/20, that use virtual memory efficiently include other hardware tricks, but the DAT and the special interrupt are the fundamental ones.

A page fault causes the processor to save its low-level state and passes control to the operating system. Depending on the type of page fault, the operating system may abort the program, or it might do the virtual memory "thing."

Doing a convincing job of imitating memory without at least enough storage to record the contents of the memory you're trying to imitate would be impossible. If you use real memory to store the contents of the memory you're imitating, you might just as well not bother doing the imitation. If you store the contents of the imitation memory on something less expensive than real memory, say disk, then you're getting somewhere. That's just what virtual memory systems do.

The memory on a system that uses virtual memory is a large file on disk. When a chunk of that memory (called a page) is referenced by a program, it is read into real memory and the processor is told that the range of virtual addresses in that page are to be found in the real addresses used to store the page. Since there is more virtual storage than real storage, every time a page is read in it bumps a page out. If the page has been modified it needs to be written back to disk. If it is unchanged, the copy already on disk is identical to it and the page need not be rewritten.

The most expensive possible memory reference in a virtual-memory system includes these steps:

* The processor detects a reference to a location that isn't available in real memory.
* The processor interrupts the instruction and jumps to a part of the operating system responsible for memory faults.
* The operating system determines that the memory the instruction wants is on disk.
* The operating system guesses which page in memory is least likely to be used soon and writes that page (if it was modified since it was last read from disk) to disk.
* The operating system reads the page required by the program from disk into the location it just cleared.
* The operating system updates the processor's table of what pages are where in real memory (DAT image).
* The instruction interrupted in the first step is resumed where it left off.

You can see that one inconvenient memory reference can involve a disk write, a disk read, and a bunch of processing.

If every memory access involved a page fault, virtual memory would slow a processor down so much nobody would use it. On a microcomputer a memory access takes something like a millionth of a second. A page fault takes about a tenth of a second to service. Slowing down processing by a factor of 100,000 isn't something to be done lightly.

A characteristic of most programs, called locality, makes it possible for most programs to run almost as fast with virtual memory as with real memory. Locality is the tendency of program execution to move relatively slowly through memory. The instructions for a subroutine are usually located in a block and the data for that subroutine is usually located in another block. Another way of looking at this is that over some short time most programs use about 20% of the memory allocated to them much more heavily than the rest.

At any moment a program has something called a working set which is the group of pages that the program is referencing frequently at that time. If the system has enough real memory to hold a program's working set and the pages in the working set change slowly, the system will need to move pages to and from the disk relatively seldom.

So, if a system has enough real memory to hold the working sets of the programs you want to run plus some memory for the operating system, it can use virtual memory without the factor of 100,000 speed penalty. In a system that is well designed for virtual memory the speed may be within a few percent of where it would be if the memory were all real.

Trouble sets in very rapidly when the amount of real memory available sinks below the working set size. Just where the critical point falls depends on the program and the algorithm the operating system uses for deciding what pages to page out, but with little enough real memory your system will start to "thrash" and you might just as well turn it off.

The thing that set me off on this was a note in the Bit Bucket of the May issue that said that Franz Lisp would be available for UniFlex soon. Lisp programs have especially large working sets. A computer with about three megabytes of real memory can run Franz Lisp comfortably. An S50 system with a limit of a megabyte of real memory running Franz would be funny in a sad kind of way. There are plenty of computers running UniFlex (and OS-9) that don't use the S50 bus, but, if you have your money in a box like mine, don't start dreaming about Franz Lisp running in virtual memory too soon.

A little story might put this in better perspective. We have a Symbolics Lisp machine at school. Symbolics designed their machines to run lisp. They have a special instruction set which practically constitutes a lisp interpreter in microcode. At lunch today another student was telling me about the warning message he got from it after a few hour's work: ***** Warning -- Only 2 Megabytes Left.

I just got a letter from Gimix telling me that they have a 68020 system with VIRTUAL MEMORY. They didn't mention a price, and even if it is low, graduate students aren't funded at a level that permits that kind of luxury. Let's imagine that my books are million sellers.

Let's also assume that Microware creates a version of OS-9/68K for the system. What will I do?

I'll go for a virtual memory system like a shot. I'd prefer one with room for a few megabytes of memory, but if that is out of the question I'll live with what I can get. If someone offers me a copy of Franz Lisp VERY cheap I might buy it, but that isn't the only reason to have a system with virtual memory.

It would be nice to be able to edit very large files without having to use "more." With virtual memory, large files could be loaded into memory. I usually move through a file pretty slowly as I edit. Global search and replace operations would page hard, but that's the price you pay.

Programs could include fancy error handling code and lots of friendly options. Good error handling can take up several times as much space as the initial program. Programs that use several megabytes come to mind that would only need a few hundred kilobytes if all error handling were removed.

I want a computer with plenty of memory, but S50 machines are limited to a megabyte. Virtual memory gives me a way out and I won't turn it down, but if I every forget that the memory is only virtual it will remind me. My disk drive will turn on its light and go shuffa-shuffa... while I catch up on my reading.

Keeping a Journal

It seems the IRS requires people who want to take their computer as a deduction to keep a log of their use. I've never been much good at that kind of thing so I built it into my computer.

At first I thought about creating a special program to keep the log, but it bothers me to do something like that just for some busybody in Washington (at least without pay). I created a quick and dirty solution, I have my password file execute a shell script before I start up. The password file entry looks something like this:

```
Peter,X,0,128,.,.,shell /H0/USAGE.LOG/cmd;
    ex shell
```

The part of the line starting with "shell" is the important part. It starts a shell which starts another shell with the file /H0/USAGE.LOG/cmd as input. When that shell script is done "ex shell" starts a fresh shell which will be left running for me.

The shell script goes like this:

```
=========================
echo Type a log entry then
     esc on a new line. >/term
chd /h0/usage.log
build temp </term
date t >temp2
merge log temp2 temp >temp3
del temp temp2 log
rename temp3 log
=========================
```

Standard input needs to be redirected because the shell (and therefore all programs it runs) is getting its input from the command file. First I prompt "Type a log entry then esc on a new line." Then I collect the response in a file called temp. I get the date and time from the system and save them in a file called temp2. Finally I add them to the end of the log file with the command sequence: merge, del, rename.

I was pretty happy with the shell script approach except for one think. The merge, del, rename method of appending a few lines to the end of a file was ugly and slow. OS-9 needs a way to append one file to the end of another.

Since I was still objecting to writing free programs for the IRS I wrote Append (which would have been easy to do in assembler) in C. With append, cmd looks like:

```
=========================
echo Type a log entry then
     esc on a new line. >/term
chd /h0/usage.log
append log </term
date t ! append log
=========================
```

This shell script looks better than the old one and it runs a great deal faster. As the year goes on and my log file gets bigger, I'll remember how glad I am that I don't have to copy it with merge each time I log on just to put a couple of new lines at the end of it.

\- - -

11 April 1985 22:46 Append.c Page 1

```
 1 #include <stdio.h>
 2
 3 main(ct,val)
 4   int ct;
 5   char **val;
 6   {
 7       char *FileName;
 8       FILE *OutFile;
 9       char c;
10
11       if(ct > 1) {
12         FileName = val[1];
13         if((OutFile = fopen(FileName, "a")) == NULL) {
14           fprintf(stderr,
               "%s: %s can't be opened for writing\n",
15             val[0], FileName);
16           exit(1);
17         }
18       } else
19         OutFile = stdout;
20
21       while((c = getchar()) != EOF) putc(c, OutFile);
22       fclose(OutFile);
23       exit(0);
24   }
```

"C" User Notes

Edgar M. (Bud) Pass, Ph.D.
1454 Latta Lane
Conyers, Ga 30207

INTRODUCTION

As already described, the C language has no I/O or operating system calls in its syntax. Rather, all such abilities are provided thru functions. This chapter discusses the standard C library functions, which provide many of the facilities available in other languages, and provide much of the power of the C language.

The "printf" function, which has been used several times in previous chapters with little explanation, is a member of the standard C library, and is discussed further in this chapter.

STANDARD C LIBRARY

The standard C library functions described below are representative of those available in a full C implementation. The C programmer must determine the structure and use of the C library for a specific implementation.

Most versions of C compilers require the programmer to include a reference to the C library with a preprocessor command similar to the following:

```
#include "stdio.h"
   and
#include "ctype.h"
   and
#include "flex.h"
   and
#include "os9.h"
```

In fact, there may be multiple C libraries in many implementations, in which case multiple "#include" commands may be required in order to specify the required libraries, the exact nature of which is highly implementation-dependent.

Some implementations of C compilers require different delimiters surrounding the name of the file to be included. Dyna-C requires no surrounding delimiters at all. The McCosh family of C compilers uses double quote symbols to indicate that the current working or data directory is to be searched. It uses "<" and ">" to indicate that the current system or "d0/defs" directory is to be searched.

In order to help the reader become oriented to reading C functions, versions of some of the simpler C library functions are provided below. Note that these are only representative and do not necessarily indicate how the function would be provided in a given implementation.

INPUT AND OUTPUT FUNCTIONS

Fclose closes a file which has been opened by fopen.

```
int fclose(fp)
FILE *fp;
```

Fgets reads a line of up to n characters, into the area pointed to by *s, from file with file pointer fp.

```
char *fgets(s,n,fp)
char *s;
int n;
FILE *fp;
```

Fopen opens a disk file for access by the C program; the name parameter should point to a null-terminated string that is the name of the file to open; the mode parameter is also a null-terminated string, and contains the access mode desired.

```
FILE *fopen(name,mode)
char *name,*mode;
```

Fprintf is similar to printf except that the formatted line is written to the file via file pointer fp.

```
int fprintf(fp,fmt,arg1,arg2,...)
FILE *fp;
char *fmt;
```

Fputs writes a null-terminated string to the file with file pointer fp.

```
char *fputs(s,fp)
char *s;
FILE *fp;
```

Fread reads binary data from the file with file pointer fp, into an area pointed to by ptr, for a length of size times number bytes.

```
fread(ptr,size,number,fp)
char *ptr;
int size,number;
FILE *fp;
```

Fscanf is similar to scanf except that the file with file pointer fp is used for input.

```
int fscanf(fp,fmt,arg1,arg2,...)
FILE *fp;
char *fmt;
```

Fwrite writes binary data to the file with file pointer fp, from an area pointed to by ptr, for a length of size times number bytes.

```
fwrite(ptr,size,number,fp)
char *ptr;
int size,number;
FILE *fp;
```

Getc returns the next sequential character from the file with file pointer fp.

```
int getc(fp)
FILE *fp;
```

Getchar is the same as getc(STDIN).

```
int getchar()
```

Gets attempts to read a line of input from STDIN.

```
int gets(s)
char *s;
```

Printf is used for formatted output to STDOUT.

```
int printf(fmt,arg1,arg2,...)
char *fmt;
```

Its function is to process the fmt string and output the arguments as directed by the fmt string. The fmt string is a null terminated string made up of arbitrary text possibly containing one or more format specifiers. There should be one format specifier for each argument in the function call. Any characters not part of a format specifier in the fmt string are passed directly to the output.

A format specifier has the following form:

%[-][w][.m]<conversion character>

where "%" signals that a format specifier is beginning, the dash (if present) changes the default right justification of the field to left justification, the "w" (if present) is the width of the field in characters, and the ".m" argument (if present) is the maximum width allowed in string conversions. If the "w" argument has a leading zero, the area of the field not filled by the conversion will be padded with zeroes. Otherwise, spaces are used to fill out a field.

The allowable conversion characters are as follows:

```
d - decimal integer format (signed)
u - unsigned integer format
x - hexadecimal notation (no leading 0x)
o - octal notation
c - single character
s - null-terminated string
e - float or double format (exponential output)
f - float or double format (decimal output)
g - e or f format, whichever is shorter
```

Putc sends a character to the file with file pointer fp.

```
int putc(c,fp)
char c;
FILE *fp;
```

Putchar is the same as putc(c,STDOUT).

```
int putchar(c)
char c;
```

Puterr is the same as putc(c,STDERR). Its output will thus normally be sent to the user's terminal even if STDOUT is redirected.

```
puterr(c)
char c;
```

Puts is the same as fputs(s,STDOUT).

```
int puts(s)
char *s;
```

Scanf is used for formatted input from STDIN.

```
int scanf(fmt,arg1,arg2,...)
char *fmt;
```

The format specified by fmt defines how an input line should be processed. Since all parameters in C are passed by value, the arguments to scanf must be pointers to objects of the corresponding types.

Ungetc pushes the character c back to the file with the file pointer fp.

```
ungetc(c,fp)
char c;
FILE *fp;
```

Ungetchar is the same as ungetc(c,STDIN).

```
ungetchar(c)
char c;
```

Unlink attempts to delete the file whose name is given by the null-terminated string pointed to by name.

```
int unlink(name)
char *name;
```

GENERAL FUNCTIONS

Abs returns the absolute value of its argument.

```
int abs(n)
int n;
{
    return (n>=0?n:-n);
}
```

Exit terminates the C program and returns to the operating system; the argument may be used as a program return code if the implementation and operating system support program return codes.

```
exit(i)
int i;
```

Max returns the larger of its two arguments. Some implementations allow more than two arguments to max.

```
int max(x,y)
int x,y;
{
    return (x>y?x:y);
}
```

Min returns the smaller of its two arguments. Some implementations allow more than two arguments to min.

```
int min(x,y)
int x,y;
{
    return (x>y?y:x);
}
```

Movmem moves a block of memory length bytes long from the address given by from to the address given by to. It is often used to move structure members, since they may not be assigned directly. It usually ensures that the move is done properly if the source block overlaps the destination block. to prevent replication. Many

implementations do not provide the movmem function.

```
movmem(from,to,length)
char *from,*to;
int length;
```

Sizeof returns the size (in bytes) of its parameter object, which may be of any type. In many implementations, sizeof is not a function, but is a part of the language, but the difference is normally not important to the user.

```
sizeof(object)
```

MEMORY MANAGEMENT FUNCTIONS

No known implementation supports all of the following memory management functions. The user attempting to port a program which calls memory management functions to a new implementation must investigate those provided by the new implementation.

Alloc attempts to return a pointer to a block of memory of nbytes bytes.

```
char *alloc(nbytes)
int nbytes;
```

Brk requests that the program's addressable data space include address.

```
brk(address)
int address;
```

Calloc attempts to return a pointer to a block of memory of n times size bytes, cleared to hex zero.

```
char *calloc(n,size)
int n,size;
```

Cdata requests that the program's addressable data space include address, in a contiguous manner.

```
cdata(address)
int address;
```

Free deallocates a block of memory previously allocated by alloc, malloc, or realloc.

```
free(block)
char *block;
```

Malloc attempts to return a pointer to a block of memory of nbytes bytes.

```
char *malloc(nbytes)
int nbytes;
```

Realloc attempts to free the space pointed to by ptr and then attempts to return a pointer to a block of memory of nbytes bytes.

```
char *realloc(ptr,nbytes)
char *ptr;
int nbytes;
```

Sbrk attempts to return a pointer to a block of memory of nbytes bytes. The space pointed to by it may not later be freed.

```
char *sbrk(nbytes)
int nbytes;
```

STRING AND CHARACTER FUNCTIONS

Inalpha tests the character c for alphabetic.

```
inalpha(c)
char c;
{
    return (isupper(c)||islower(c));
}
```

Isdigit tests the character for being within the range ('0'...'9').

```
isdigit(c)
char c;
{
    return (c>='0'&&c<='9');
}
```

Islower tests the character for lower case letters.

```
islower(c)
char c;
{
    return (c>='a'&&c<='z');
}
```

Isspace tests the character for blank, newline, or tab.

```
isspace(c)
char c;
{
    return (c=' '||c='\n'||c='\t');
}
```

Isupper tests the character for upper case letters.

```
isupper(c)
char c;
{
    return (c>='A'&&c<='Z');
}
```

Atoi converts the given string, which is assumed to be a sequence of digits ('0'...'9'), into a binary integer representation.

```
int atoi(s)
char *s;
```

Itoa converts its integer argument into a character string pointed to by s.

```
int itoa(n,s)
int n;
char *s;
```

Reverse reverses the characters of the null-terminated string pointed to by s.

```
reverse(s)
char *s;
{
    char *b,*e,c;
    for (b=s;e=b+strlen(s);*b)
        {
            c=*b;
            *b++=*--e;
            *e=c;
        }
}
```

Sprintf is similar to printf except that the formatted line is placed into the string pointed to by line.

```
sprintf(line,fmt,arg1,arg2,...)
char *line,*fmt;
```

Sscanf is similar to scanf except that the string pointed to by line is scanned to do the input conversions.

```
sscanf(line,fmt,arg1,arg2,...)
char *line,*fmt;
```

Strcmp compares its two null-terminated argument strings s and t, and returns a negative value if s < t, zero if s == t, a positive value if s > t.

```
int strcmp(s,t)
char *s,*t;
{
    while (*s==*t)
        if (*s++=='\0')
            return (0);
    return (((int)*s)-((int)*t));
}
```

Strcpy copies the null-terminated string pointed to by t into the string pointed to by s.

```
strcpy(s,t)
char *s,*t;
{
    while (*s++=*t++);
}
```

Strlen returns the length of the null-terminated string pointed to by s, not including the null character.

```
int strlen(s)
char *s;
{
    for (p=s;*p;++p);
    return (p-s);
}
```

Strsave attempts to make a copy of the null-terminated string pointed to by s into an area of memory obtained by a call to alloc.

```
char *strsave(s)
char *s;
{
    char *p,*alloc();
    if ((p=alloc(strlen(s)+1))!=NULL)
        strcpy(p,s);
    return (p);
}
```

Tolower converts upper case letters to lower case; it converts other characters to themselves.

```
char tolower(c)
char c;
{
    return (isupper(c)?(c+32):c);
}
```

Toupper converts lower case letters to upper case; it converts other characters to themselves.

```
char toupper(c)
char c;
{
    return (islower(c)?(c-32):c);
}
```

SUMMARY

This chapter completed the tutorial begun in the previous chapter. If you have access to a C compiler, use it. If you do not have a copy of "The C Programming Language" by Kernighan and Ritchie, buy it. Enter and run several of the programs in K & R.

EXAMPLE C FUNCTIONS

Following is a set of example C functions which illustrates the use of pointers, addresses, and several other concepts. It also illustrates the fact that C programs may be written very implementation-dependent. At this point, the reader should be able to understand every point of the language used by this set of functions.

```
/*
**
**      Formatted print functions
**
**      These routines are tricky since they may be
**      called with varying numbers of arguments.
**      They depend on the fact that this C compiler
**      pushes arguments in the order of occurrence.
**      Thus the last argument in the list is nearest
**      the top of the stack (lowest address).
**
**      These versions are non-standard in that they
**      require the number of parameters to appear as
**      the last parameter.
**
*/

/*
**      Format and print to standard output
*/
printf (a, n)
int     *a, n;
{
    int     *fmt;
    char    buf[140];

    fmt = &a + n;
     fmt (fmt, *fmt, buf);
    return (fputs (buf, stdout));
}

/*
**      Format and print to an i/o stream.
*/
fprintf (a, n)
int     *a, n;
{
    int     *fmt;
    char    buf[140];

    fmt = &a + n;
     fmt (fmt, *fmt, buf);
    return (fputs (buf, fmt[1]));
}

/*
**      Format into memory at the address given.
*/
sprintf (a, n)
int     *a, n;
{
    int     *fmt;

    fmt = &a + n;
    _fmt (fmt, *fmt, fmt[1]);
}

/*
**      workhorse function.
*/
 fmt (argptr, format, buf)
int     *argptr;
char    *format,
        *buf;
{
```

```
char    c,
        padchr,
       *tstr,
        tbuf[30];
bool ljust, zpad, tsfull;

int     i,
        padlen,
        len,
        prec,
        fldwidth;

while (c = *format++) {
    if (c == '%') {
        if (ljust = ((c = *format++) == '-'))
            c = *format++;
        if (zpad = (c == '0')) {
            padchr = '0';
            c = *format++;
        }
        else
            padchr = ' ';
        for (fldwidth = 0; isdigit (c);
                c = *format++)
            fldwidth = fldwidth * 10 + c - '0';
        if (c == '.') {
            prec = 0;
            while (isdigit (c = *format++))
                prec = prec * 10 + c - '0';
        }
        else
            prec = 10000;
        tstr = tbuf;
        tsfull = TRUE;
        switch (c) {
            case 'd':
                itoa (*--argptr, -10, tstr);
                break;
            case 'x':
                itoa (*--argptr, 16, tstr);
                break;
            case 'o':
                itoa (*--argptr, 8, tstr);
                break;
            case 'u':
                itoa (*--argptr, 10, tstr);
                break;
            case 'b':
                itoa (*--argptr, 2, tstr);
                break;
            case 'c':
                *buf++ = *--argptr;
                tsfull = FALSE;
                break;
            case 's':
                tstr = *--argptr;
                break;
            default:
                *buf++ = c;
                tsfull = FALSE;
                break;
        }
        if (tsfull) {
            if ((len = strlen (tstr)) > prec)
                len = prec;
            if ((padlen = fldwidth - len) < 0)
                padlen = 0;
            if (ljust)
                buf = strncpy (buf, tstr, len);
            for (i = 1; i <= padlen; i++)
                *buf++ = padchr;
            if (ljust == 0)
                buf = strncpy (buf, tstr, len);
        }
    }
    else
        *buf++ = c;
}
*buf = NULL;
}
```

68000 User Notes

Phillip Lucido
2320 Saratoga Drive
Sharpsville, Pa 16150

Things have picked up since last month, so I am not at a loss for subjects. For one thing, I now have a computer with a full megabyte of RAM, with a total upgrade cost of about $210. For another, I have a new upgrade of OS-9/68K, which seems to have reached a stable state, so I can now get to work on the C compiler I've had for several months but have been unable to use.

MacWait Some MacMore

One thing hasn't changed since last month. I am still without the **Inside Macintosh** manual, even though it was ordered six and a half weeks ago. Not to fear, though! I've managed to track down the reason for the delay. It seems that Apple reduced the price for IM from $100 to $25. People who sent in $100, but haven't yet received their books, will instead have their money returned, along with a note to resubmit the order with a check for the new price. Why not just send a book plus $75? Ya got me!

Anyway, I'm not about to wait for Apple to get around to refunding my $100. As soon as I heard about the reduced price, I sent in a new check. Supposedly, orders are now being filled with only a two day turnaround. If that's true, then I should have my copy of IM in another day or two. By next month's column, I should have something halfway intelligent to say about programming the Mac.

The address to order your copy of **Inside Macintosh** is:

Apple Computer Inc.
467 Saratoga Ave.
Suite 621
San Jose, CA 95129

This is the address for IM orders only! The books now being shipped are just photocopied sheets. If you're not in a hurry, you might wait for the published version, though I'm not sure when that will be available.

Mega Mega Mega!

As I mentioned last month, the price for 41256-type 256K dynamic RAMs has recently fallen precipitously. I just bought 34 chips (2 extras for spares) at $6.25 a chip, or $212.50 plus shipping for a full megabyte

of memory. By the time this column appears, you should probably be able to find them for less than five bucks a chip.

Performing the upgrade was not as simple as plugging the 41256s in place of 4164s in my memory board. I have a Hazelwood DM-256 board, which was not planned with the upgrade in mind. Fortunately, the wiring changes are not too extensive. For those of you who might be interested in doing your own upgrade, this is how to go about it.

First, it is a good idea to mark the changes directly on the schematic of the DM-256, just so you will know exactly what has been done to your board. You should have the schematic and the board layout sheet so you can translate the chip numbers I will use to the actual chips on the board. Also, be extremely careful when making the wiring changes. I tack soldered the required wires on the back side of the board, and it is very easy to accidentally form a solder bridge to an adjacent pad or line.

The major board change involves wiring pin 1 of the RAM chips together. Daisy chain a wire on pin 1 of each row of RAMs. The four rows consist of U30 to U37, U38 to U45, U46 to U53, and U54 to U61. Each row is individually wired, and is not connected to the other rows (yet). Rather than wiring pin 1 between each chip in a row using 7 small wires, I used one large wire, and carefully stripped the wire in 7 different places, moving the insulation apart to give me enough bare wire to wrap around each socket's pin 1.

The other wiring additions are as follows. To connect the RAM pin 1 wires to the address line multiplexors, wire U13 pin 15 to U30 pin 1 and U38 pin 1, and wire U20 pin 15 to U46 pin 1 and U54 pin 1. The multiplexors are wired to the address lines by wires from U18 pin 7 to U13 pin 12 and U20 pin 11, and from U18 pin 4 to U13 pin 11 and U20 pin 12. Wire U5 pin 14 to U5 pin 15, changing the board selection logic. Finally, U13 pin 9, U13 pin 10, U20 pin 9, and U20 pin 10 must all be connected to a convenient ground, such as U13 pin 8 and U20 pin 8.

Only one line must be cut on the board. Find the trace from U18 pin 7 to U5 pin 15, and cut it by scratching through the line with a knife. Finally, set switches 1, 2, and 7 open, and switch 8 closed. Open switches 3 to 6 unless you have a system capable of handling over a megabyte, in which case these switches address the board to the proper megabyte.

A warning - this upgrade should obviously be undertaken with much care. It is best if you can understand the reason for the changes I made for yourself, just so you can be sure that I made no errors in my description of the upgrade.

I was lucky. After making the changes, the board worked properly the first time. I booted up OS-9/68K, and was told that 1012K of RAM was present. It seems that my CP-08 68008 board hides 12K of the RAM, which is therefore not available to the system. I have no memory test programs, but since loading an OS-9 module involves a CRC calculation to check module integrity, I simply loaded the memory with many programs to make sure the RAM was working.

As you might expect, a full megabyte of RAM is going to be very useful. RAM disk drivers are included in OS-9/68K, so I reconfigured my system with a 256K RAM disk. I then changed my startup file to move everything from my DEFS, LIB, and SYS directories to the RAM disk, and to load many commonly used utilities, the screen editor, and the various C compiler phases into RAM. After all this, I still have about 350K free, and the boot only takes about a minute. C compiles are now done with temporary files written to the RAM disk, so the only time the hard disk is accessed during a compile is when the source file is read and the final object file is written, resulting in C compiles that take only half a minute.

Presenting the NEW OS-9/68K

Version 1.1 of OS-9/68K (OSK for short) is now out. If you're still running an old version, get the update. This new version is more stable and bug-free than previous ones, with several new utilities and features. I'll go through some of the changes here. Some of these were first implemented under version 1.0, but I didn't do much of anything with that version, since the C compiler that I have (version 1.2) only works with OSK version 1.1.

First, the formats of various internal headers and tables has completely changed. Module headers are now a minimum of 48 bytes long, with many bytes reserved for future expansion. For instance, there is a module

field reserved as an offset to a symbol table, to be used in a future symbolic debugger.

Microware has continued adding new utilities. Perhaps the most useful is **make**, which I have been wanting for a long time. This is a program that automates the production of large software packages, that I have mentioned before when discussing Unix. It is a very complete version, implementing many of the abilities of the Unix version, plus others useful in the OSK environment. The screen editor, **scred**, has been improved, with a new setup program to simplify configuration of the editor for a new terminal.

An Electronic Stopwatch

Now that I have a version of OSK matching my version of the C compiler, I finally managed to put together a useful program for this column. The C program, **time**, is patterned after the Unix utility of the same name (surprise, surprise). It will run another program, displaying the total execution time, time spent in the system state, and time spent in the user state. Time is spent in the system state while a process is busy executing an OS9 trap, and in the user state when executing code outside of these traps. Time spent waiting for I/O or other resources will not be counted in system or user times, but will affect the total execution time.

Time is made possible by several entries in the OSK process descriptor. The process descriptor is an internal OSK table that holds all of the necessary information for controlling an executing process. This is the data table from which the **procs** utility derives its information. The entries that are used by **time** record the time when a process started, the number of ticks spent in the system state, and the number of ticks spent in the user state.

There are more new entries in the process descriptor that hold useful information, even if they don't really have anything to do with the execution time. Thus, as an option, **time** will display the number of F$ and I$ OS9 calls made by a process, and the number of characters read and written. These are the same extra data items available in the new **procs** command.

Time is easy to run. Instead of typing a normal shell command, just prefix the command line with "time ", or "time -e " if you want the extra information. **Time** will run the command, then examine the process descriptor when the command is finished, and print its summary.

Time manages to stick to the existing OSK conventions for utility programs. An option is specified by preceding it with a dash, and an option of '-?' will cause a short usage summary to be printed in the normal format established by the Microware utilities. I consider these conventions to be very important, since some uniformity among programs can make working under OSK a lot smoother.

There are some limitations to **time**. Since the command is forked directly, instead of calling the shell, **time** cannot handle procedure files, only executable programs. Also, the times reported are only those of the forked command. If that command itself forks to other modules (like cc), the sub-module times will not be included. This is a problem with OSK, and **time** can't do anything about it.

Dissecting Time

There are a few interesting points to be examined in **time**. For one thing, the command line arguments must be specially handled. Normally, the command line is split into individual arguments before being passed to **main()**. In addition, special characters like quotation marks are processed and removed. In **time**, though, we don't want the command line to be split into individual arguments, since the **os9fork()** function wants a single string with command name and arguments all together. Neither do we want any special processing of characters in the command line.

Under the current C compiler and OSK, the command line processing is done by a routine called **_initarg**, which is called from the C startup module, cstart.r. The default **_initarg** is found in the C library, and can therefore be replaced by defining a new version in the object code before the library is searched. This is done in **time**. The routine, which must be written in assembly language, is passed a pointer to the command line in A0, with a pointer to the module name on the stack. It should create the normal **argc** and **argv** parameters on the stack, in preparation for calling **main()**. The replacement routine just returns an argument count of two. The

first argument is the module name, as normal, and the second is the command line, with no special processing done.

There is another assembly language routine required, to use the OS9 service request **F$GPrDsc**, that copies a process descriptor into a program's buffer for examination. This routine is straightforward, using the Microware C conventions for argument passing so it can be used by C code.

Routines **main()** through **getarg()** take care of doing what processing of the command line is required. The start of the command line is checked for any options. The name of the command to be executed is then pulled from the command line start. The remainder of the line will be passed as the parameter string when forking to the command.

The real work is done by **exec()** and **pstat()**. **Exec()** first sets up a signal handler. This handler will send any signals intercepted by **time** on to the command being executed. Next, the command is begun, with a call to **os9fork()**.

After the command has started, **time** must wait for it to finish. A simple call to **wait()** does not work, since **wait()** would return after the command has completed and its process descriptor is freed. Instead, **time** repeatedly sleeps for half a second, and checks the process descriptor after each sleep to see if the command is finished. When it is, the results are output, and **wait()** is finally called, allowing the process descriptor to be released.

The routine **pstat()**, which prints the results, reads the current time. Note that the **_sysdate()** call to do this differs from that in the C manual, which is inaccurate. Since the starting time, from the descriptor, and the ending time are both in Julian time, time elapsed is simple to calculate. The **_sysdate()** call also returns the number of ticks per second, so the system and user ticks from the descriptor can be converted to seconds.

Enough on how it works. The only thing left is actually compiling it. The command to do this is "cc time.c -ix". This causes the final code to use the **cio** and **mathi** trap packages, so the code length is under 3000 bytes, instead of over 10000, which is the result if the trap packages are not used.

Enough Already

This is probably the longest column I've had yet, especially with the listing. I'll stop here. Back next month with more on the new OS-9/68K and C, and with any luck, a copy of **Inside Macintosh**. (Late Flash - no book, but I did get my $100 check back. At least Apple knows I exist.)

```
/*
    time.c

    Run a program and print the execution time statistics
    when finished.

    Syntax: time [<opts>] <shell cmd>
    Options:
            -e      Print extra information

    Prints the real, system, and user execution times.
    With the -e option, extra information about the number
    and type of OS9 calls made and number of bytes read
    and written is printed.

    Philip Lucido, April 10, 1985
*/

#include <stdio.h>
#include <module.h>
#include <procid.h>

#define TRUE    1
#define FALSE   0

/*
    Assembly language routines.  Some assembly language is
    required to talk directly to OSK at a lower level than
    normal to C.
*/

#asm
```

```
* Replacement routine for argument processing to argc and argv.
* Returns argc = 2, argv[0] = module name, argv[1] = entire
* parameter string.
_initarg:   movea.l (a7)+,a5        return address to a5
            move.l  (a7)+,d0        pop module name pointer
            movem.l d0/a0,-(a7)     create argv[0..1] on stack
            pea     (a7)            argv = address of array
            pea     2               argc = 2
            jmp     (a5)            return to caterr

* Interface for F$GPrDsc, which copies a process descriptor
* into a buffer for inspection.  Calling sequence from C is:
*   int _gprdsc(pid,length,buffer)
*       int pid;        /* process id */
*       int length;     /* number of bytes to get */
*       procid *buffer; /* process descriptor buffer pointer */
* Returns 0 if successful, -1 if error, with error code in
* 'errno'.
_gprdsc:    movem.l d1/a0,-(a7)     (is this necessary?)
            movea.l 12(a7),a0       buf addr - d0/d1 already set
            os9     F$GPrDsc        doittoit
            bcs.s   gprdsc1
            moveq   #0,d0           success - no error
            bra     gprdsc2
gprdsc1     move.l  d1,errno(a6)    error - save error code
            moveq   #-1,d0          return indication of error
gprdsc2     movem.l (a7)+,d1/a0     restore and return
```

```
#endasm

/*
    Declare program global variables
*/

int opte;       /* -e - display extra information */
int child;      /* process ID of executing child */
procid prd;     /* process descriptor table for child process */

/*
    Start of program
*/

main(argc,argv)
    register int argc;
    register char **argv;
{
    register int i;
    char *paramstr;
    register char *arg;
    char *getarg();

    paramstr = argv[1];
    for (;;) {
        arg = getarg(&paramstr);    /* try to get command name */
        if (*arg == '\0')
            exit(0);                /* no command - quit */
        if (*arg != '-')
            break;                  /* command name gotten */
        option(arg);                /* else option argument */
    }
    exec(arg,paramstr);             /* do actual work */
}

/*
    Process an option argument
*/

option(p)
    register char *p;
{
    register char c;

    while (c = *++p)
        switch (c) {
        case 'e':
            opte = TRUE;
            break;
        case '?':
            usage();
            exit(0);
        default:
            exit(_errmsg(1,"unknown option '%c'\n",c));
        }
}


/*
    Print the program usage summary
*/

usage()
{
    fprintf(stderr,"Syntax: time [<opts>] <cmd>\n");
    fprintf(stderr,"Function: Time the execution of a program\n");
    fprintf(stderr,"Options:\n");
    fprintf(stderr,"\t-e\t\tPrint extra information\n");
}

/*
    Extract the next argument from the argument string,
    terminate it, and update the pointer to the remainder of
    the string.  Returns the next argument pointer.

    Simple version - Skip initial blanks and tabs, find next
    blank or tab, terminate argument, and skip trailing blanks
    and tabs.  Does no special processing of quotes, commas,
    or backslashes.
*/

char *getarg(pp)
    register char **pp;
{
    register char *p1;
    register char *p2;
    register char c;

    p1 = *pp;
    while ((c = *p1) == ' ' || c == '\t')
        ++p1;
    p2 = p1;
    while ((c = *p1) != '\0' && c != ' ' && c != '\t')
        ++p1;
    *p1++ = '\0';
    while ((c = *p1) == ' ' || c == '\t')
        ++p1;
    *pp = p1;
    return p2;
}

/*
    Do the actual work.  Start the program, wait for it to die,
    and print the timing info when it does.  If the child
    process returns an error, then this function exits with the
    same status.
*/

exec(name,param)
    register char *name;
    register char *param;
{
    int sighandle();        /* signal intercept handler */
    unsigned status;        /* returned status from child */

    intercept(sighandle);           /* intercept signals */
    child = os9fork(name,strlen(param)+1,param,1,0,0,0);
    if (child == -1)



    do {        /* main loop - wait for program to die */
        tsleep(0x80000080);         /* sleep for 0.5 seconds */
        if (_gprdsc(child,sizeof(prd),&prd) == -1)
            exit(_errmsg(errno,"can't read process descr - "));
    } while (prd._queueid != '-');
    patst();                        /* print the statistics */
    wait(&status);                  /* allow child to go away */
    if (status)
        exit(_errmsg(status,"error in child process - "));
}

/*
    Signal handler - If child started, then just pass the
    signal on.
*/

sighandle(signal)
    int signal;
{
    if (child > 0)
        kill(child,signal);
}

/*
    After child process is complete, print the timing data,
    and other data if requested to by option -e.
*/

patst()
{
    unsigned date, time, tick;
    short day;
    register unsigned real;     /* time since start, in secs */
    register unsigned tps;      /* ticks per second */

    /* WARNING - _systime() call differs from manual */
    _systime(3,&time,&date,&day,&tick);
    tps = tick >> 16;
    real = (date - prd._datbeg)*86400 + (time - prd._timbeg);
    fprintf(stderr,"\nReal          %9d\n",real);
    fprintf(stderr,"User          %12.2f\n",
                        (double) prd._uticks / tps);
    fprintf(stderr,"System        %12.2f\n",
                        (double) prd._sticks / tps);
    if (opte) {
        fprintf(stderr,"F$ calls      %9d\n",prd._fcalls);
        fprintf(stderr,"I$ calls      %9d\n",prd._icalls);
        fprintf(stderr,"Bytes read    %9d\n",prd._rbytes);
        fprintf(stderr,"Bytes written %9d\n",prd._wbytes);
    }
}
```

ADA[R] And The 68000

BY
THEODORE F. ELBERT
THE UNIVERSITY OF WEST FLORIDA
PENSACOLA, FLORIDA 32514

Part 3

SOFTWARE ENGINEERING ASPECTS of the ADA LANGUAGE

In Part 2 of this series, several general design goals of software engineering were introduced. These general design goals are:

- modifiability
- reliability
- portability
- efficiency
- understandability
- reuseability.

Each of these goals is discussed below, and the language features that promote achievement of that goal in an Ada program are identified.

Modifiability. As the name implies, this property refers to ease with which controlled change may be incorporated into existing software. Embedded computer systems in particular have a need for software with this property, because of the many modifications made to operational software over the life cycle of the system. Important to achievement of modifiability are

- controlled change: the absence of side effects that cause unexpected modification of program behavior.
- self-documentation: the software itself should reflect the underlying algorithmic design.
- management of complexity: software modifications should not increase the complexity of the solution represented by the software.:

Typically, modification to software is often performed by programmers other than those involved with the original design, and it may be performed at a site other than that at which the original design was developed. The Ada language features of strong typing, generic units, packages, and separate compilation serve to enhance the modifiability of software.

Efficiency. The term efficiency, in this context, refers to use of hardware resources and execution time by compiled code. While compiler efficiency -- referring to the use of time and resources by the compiler -- may be of some interest, the embedded system programmer is much more concerned with the execution characteristics of compiled code on the target machine. In the embedded system environment, optimization with respect to either execution time or with respect to computational resources may be required. Furthermore, a single program may require execution time optimization in one part of

the program and resource optimization in another. In the development of the Ada language, any proposed feature for which efficient implementation could not be clearly envisioned using standard compiler design techniques was rejected.

Reliability. The general concept of reliability -- as applied to software -- concerns the question of whether the software will perform its intended function accurately under normal conditions, and whether it will do so consistently. Abnormal conditions should never result in erroneous performance masked as proper performance. Instead, the software should be able to recover from such conditions or, at the very least, continue execution with reduced effectiveness -- a property known as graceful degradation. Thus, the software should avoid catastrophic failure which results in program abandonment by the underlying operating environment.

The reliability of software is dependent upon the correctness of the design specifications, upon the correctness of the mapping of the design to the software implementation, and upon the ability of the software to recover from exceptional conditions if they should arise. For large programs, it is often difficult or impossible to produce a requirements specification which is complete and invariant, simply because of the sheer complexity of the problem. Furthermore, in the case of embedded systems, the hardware design may be taking place concurrently with the software design, or the system operational requirements may be in a state of almost constant change. In this situation, the correctness of the design is a nebulous concept, so that one can never be sure of absolute reliability, even if the mapping of the design to the software implementation is known to be correct. A realistic definition of software reliability is[1] "that a program should meet its specifications, should never produce 'incorrect' output regardless of the input, should never allow itself to be corrupted, should take meaningful and useful action in unexpected situations, and it should completely fail only when further progress is completely impossible."

To achieve a high level of reliability in software nearly always requires a reduction in code efficiency, since features such as data checking and exception handing require a large amount of extra code and usually entail an increase in execution time. It is generally conceded, however, that reliability is cost-effective in the modern embedded system application. Some of the reasons for this concession are

- The cost of system failure could be immense, not only in terms of equipment failure, but also in terms of human lives.
- The tendency today is towards increased hardware capability at lower cost and smaller size.
- It is much more difficult to improve an unreliable system then it is to improve an inefficient system.
- Inefficiency is immediately apparent. Reliability problems are much more insidious, and therefore are more capable of causing damage.
- As a market consideration, unreliable software will not sell, regardless of the efficiency.:

In later parts of this series, the reader will see several features of the Ada language which promote reliability in Ada programs. Language features such as strong typing and separate compilation tend to ensure the correctness of the mapping from the design to the software, while the exception handling capability of the language permits a program to recover from unexpected situations.

Understandability. The understandability of software relates to the extent that an observer can grasp the functional aspects of the software, and the relationship between the software and other system components. In order for software to be understandable, its very form must imply to an observer the structure of the design being implemented. This concept extends well beyond the mnemonic use of meaningful data names and of ample commenting. It includes the use of a meaningful coding style, the use of an appropriate structured design methodology, careful selection of how data is structured, and the incorporation of other features that respect the role of programming as a human activity. The ease of maintenance of a program obviously depends upon the property of modifiability -- that change can be incorporated easily and without harmful side effects of an increase in complexity --

and the property of understandability -- that the manner in which the design is mapped to the software implementation is easily comprehended.

The understandability of the software implementation of a design can be highly dependent upon the implementation language. Since one of the primary concerns of the Ada language development effort was the realization that programming is a human activity, one finds in the language many features that can serve to enhance the understandability of Ada software. The strong typing, the modern control structures, and the data abstraction features of the language are examples.

Portability. Portability refers to the ease with which software can be moved from one computer to another. In an era of rapidly escalating software costs, it seems only reasonable to incorporate features that enhance the portability of the software design, even at the expense of increased development costs. The idea of portability is not new, of course, since any high order language is portable to some extent. Only assembler programs are perpetually tied to specific hardware. The degree of portability differs among high level languages, and depends upon the particular implementation of the language on the underlying machine. In the embedded system environment in particular, a program written in a high level language often must depend upon assembler language subprograms to exploit some feature of the underlying hardware, or upon the underlying operating system for features like task synchronization and exception handling. Current practice is to enhance the portability of programs by isolating those sections of code that are dependent upon either the underlying hardware or the underlying operating system. The most prevalent method of achieving this isolation is through the use of replaceable subprograms. Moving of the software from one machine to the other then requires replacement only of these subprograms.

A high degree of portability is inherent in all programs written in Ada, because they are virtually independent of the underlying hardware and operating system. Access to features of the underlying hardware and of the operating system is provided in the Ada language itself. The burden of providing this access is borne by the Ada compiler and run-time support software. Furthermore, the Ada language itself has been protected against the use of subsets or extensions by the fact that the name Ada has been copyrighted. Before any compiler can be called an Ada compiler, it must be validated by the Department of Defense; such validation requires that a compiler successfully pass a battery of some two thousand different tests designed to validate conformance with the language specifications.

Not only do these features enhance program portability, they also ensure that programming skills are portable from one working environment to the other -- an important consideration to the highly mobile professionals in the defense industry. To further enhance the commonality of work environments, a standard Ada Programming Support Environment (APSE) has been specified, so that the programmer is provided not only with a standard language, but also with a standard set of software development tools.

Reuseability. While portability refers to the ease with which software can be moved from one machine to the other, the term reuseability is used to describe how well software components lend themselves to use by independent programs. The concept of reuseable software is not new by any means. For some time, programmers have made use of subprograms called from standard support libraries to achieve the implementation of commonly used algorithms. In the development of reuseable subprograms, care must be taken to preserve the integrity of the calling environment, and to provide an efficient interface between the subprogram and the calling environment. But the concept of reuseability can be extended beyond the reuse of a subprogram. Such things as data structures, groups of physical constraints, or data itself can be reviewed as reuseable resources. In Ada, the concept of reuseability is given wide applicability through the package and generic unit features of the language.

NEXT: Ada's Program Units.

[1]Ross, D. T., Goodenough, J. B., and Irvine, C.A., "Software Engineering: Process, Principles, and Goals, Computer 65: (May, 1975).

- - -

Basic OS-9

by Ron Voigts

A DISK IS BORN

Up to a few years ago I considered a computer disk to be a convenience. It would store programs and save data for later use. What else could you say for it? If you initialized one disk you initialized them all. Although the systems I had worked on varied, creating a diskette was about the same. Usually some command was used like INIT or FORMAT. The disk drive would take off clicking away, marking off where track and sector information was to be located. A directory was written on one of the tracks to hold the names and location of information on the disk. And that's all there was, until OS-9 came along.

The first time I initialized an OS-9 disk, I figured, "Well here we go again!" I put the disk in drive /D1 and typed:

FORMAT /D1

The drive came on as before and started to click away. Then it stopped and replied:

COLOR COMPUTER DISK FORMATTER 1.2
FORMATTING DRIVE /D1
Y (YES) OR N (NO)
READY?

Well this seemed like a reasonable request. So I replied with a "Y" and away we went. Before long it stopped and asked:

DISK NAME:

At that moment I felt very personable with the disk. It was no longer a face in the crowd; or rather, a disk in a box. My disk was an individual. It had a name!

The best way to find out what is on that circular piece of plastic is to let OS-9 show you. There is a nice little command in you commands directory called "DUMP". DUMP will list to your display a file's contents in both hexadecimal numbers and ascii characters. If a particular byte can not be represented with an ascii character, it shows a dot to remind you something is there. Entering:

DUMP MY_FILE

will output a file called MY_FILE to the terminal. Tack on a ">/P" and the output will go to the printer. A nice feature is that DUMP can be made to treat the disk as a file. Usually a disk drive is recognized by a /D0 or a /D1, or something like that. If a @ is added on, the disk in that drive can be treated as a file. So if you enter:

DUMP /D0@

the disk will be dumped to the terminal and look something like this:

```
      0 1 2 3 4 5 6 7  0 2 4 6
ADDR  8 9 A B C D E F  8 A C E
====  +-+-+-+-+-+-+-+- + + + +
0000 00027612004F0001 ..v..O..
0008 0000020000FF4950 ......IP
0010 0200120000000008 ........
0018 303253051F0C3A43 02S...:C
0020 4F4C4F5220434F4D OLOR COM
0028 5055544552204449 PUTER DI
0030 53CB000000000000 SK......
0038 0000000000000001 ........
```

What you are looking at is the birth certificate of one of my disks. Officially this is known as the **Disk Identification Sector**. It is Logical Sector Number 0 on the disk, or LSN 0 for short. The **Logical Sector Number** is the way that sectors are referenced, instead of referring to the track and sector number. If there are n sectors on a disk, the LSN's range from 0 to n-1. On my system, disks have 630 sectors, so I can access LSN 0 through LSN 629.

Looking at the bunch of hex numbers for LSN 0 on my disk I dumped earlier can make you a bit dizzy. It is easier to map them out to better understand their purpose. Here is a breakdown on my disk:

ADDR	SIZE	NAME	MY DISK
0000	3	sector total	000276
0003	1	track size	12
0004	2	map size	004F
0006	2	cluster size	0001
0008	3	root start	000002
000B	2	owner	0000
000D	1	attributes	FF
000E	2	identification	4950
0010	1	format	02
0011	2	sectors/track	0012
0013	2	reserved	0000
0015	3	bootstrap file	00000B
0018	2	bootstrap size	3032
001A	5	creation time	
		Y:M:D	53051F
		H:D	0C3A
001F	32	name	43toCB
		COLOR COMPUTER DISK	

Much of the information stored in LSN 0 is specific to the particular disk. Is

it single sided or double sided? Is it 5" or 8"? Maybe it's a hard disk! When was it created? How large is the disk? It's best if I give you an interpretation for my disk.

First, this table shows that my disk has $276 (630) sectors on it. (All the numbers in the table are in hexadecimal. I'll use them for our analysis, but put their decimal equivalent in parentheses.) There are $12 (18) tracks on my disk. The sector allocation map contains $4F (79) bytes showing which sectors are used. There is 1 sector per cluster on this disk, as is the case on most disks (some large storage devices, such as a large Hard Disk or High-speed Tape System, may use 2, 4, or more, sectors per cluster). A cluster is the **smallest** increment of storage assigned on a disk. The root directory's file descriptor is at LSN 2. The disk's owner is user #0000 (that's me!). It's attributes are $FF which translate into:

D S PE PW PR E W R

The disk identification number is a "random" number so that the system can tell if an unexpected disk swap has occurred. The format byte is concerned only with the last three bits. They are:

BIT 0 - number of sides
0 = one
1 = two
BIT 1 - density
0 = single
1 = double
BIT 2 - track density
0 = 48 TPI
1 = 96 TPI

The format number on my disk is $02. We're only concerned with the last three bits, which are:

0 1 0 (binary 2)

This means I have a single-sided, double density, 48 TPI disk. There are $12 (18) sectors per track. Don't worry about the reserved 2 bytes. They're not used for anything. My bootstrap file is at LSN $0B and is $3032 bytes long. If the bootstrap's pointer had been 0, it would mean that this wasn't a bootable disk. 5 bytes are used for the disks creation and time. And finally comes the disk's name. It can be a maximum of $20 (32) characters long. Whew, that's a lot of information!

Now You know everything about your disk. Right? Well, actually we've only scratched the surface. Another important aspect is the allocation of disk space. Remember the "map size" in LSN 0 at $0004. Well the map is located at LSN 1. Here is a DUMP of my disk starting at address $0100 which is the beginning of LSN 1.

```
      0 1 2 3 4 5 6 7  0 2 4 6
ADDR  8 9 A B C D E F  8 A C E
====  +-+-+-+-+-+-+-+- + + + +
0100  FFFFFFFFFFFFFFFF ........
0108  FFFFFFFFFFFFFFFF ........
0110  FFFFFFFFFFFFFFFF ........
0118  FFFFC7F8080FFFFB ..Gx...{
0120  FFFFF87EFFE0D9FF ..x..`Y.
0128  0FFFFFFFFFFFFFFF ........
0130  FFFFFFFFFFFFFE3C .......<
0138  21FC0000000007FB !|.....{
0140  0000000000000000 ........
0148  000000000FFFE3FF ......c.
0150  FFFFFFFFFFFFFFFF ........
```

This shows which clusters are used and which are not. (And you thought this was going to get easier!) Actually the map is not difficult to read. If you remember from before, my map size is $49 bytes long. So this map starts at $0100 and ends at $014E. The rest of LSN 1 is filled with $FF's. (The $FF's do have some meaning. My disk is 35 tracks long. If this disk were put in a drive that could read 40 tracks and the "RS format", it could still be read. The trailing $FF's tell the system that the sectors beyond track #34 are used. In reality they don't exist. There's no chance the 40 track drive will try to use them.) I didn't print the rest of LSN 1 since I thought that would be boring.

The trick is to learn how to read this. It's very easy. Every byte represents 8 clusters (which, on this disk, is 8 sectors). At address $0100, LSN 0 thru LSN 7 are represented. The next byte is for LSN 8 through LSN 15. If a bit is set in any byte then the LS is being used. If it's cleared, the LS is available. As you can see, the byte at $0100 and $0101 have $FF's in them. The $FF means all 8 bits are set, so all of these LS's are used.

Maybe to make it a "bit" clearer (pun intended), we can jump down to address $0122, which holds $C7. Now in binary this looks like:

1 1 0 0 0 1 1 1

This represents LSN 200 thru LSN 207. Reading the map from left to right, 200 and 201 are used, 202 to 204 are clear, and 205 to 207 are used. Pretty easy, huh? Trying to read it may not be the easiest thing, but every time you write to the disk this map is read and updated by OS-9 while you sit back knowing your program is being saved!

The OS-9 directory, /D0/CMDS, contains a command called "FREE". FREE will tell you the disk's name and its available memory. For more information you can use a program like the one at the end of this months column. It's called "DISKID". The program was designed to give you a better idea of what is on the disk. I tried to incorporate most of the information in LSN 0. In some cases the output is the actual number; in others it is an interpretation of the information.

There are 4 parts to this month's program. The first part initializes the necessary parameters. A complex variable type is created to hold the information from LSN 0. The second part reads LSN 0 into id which was created by the complex variable. The third part does the actual analysis of the information in variable id. I left the index of bytes in array id.b in hexadecimal format since it is easier to the associate it with the map given earlier. The output is printed in decimal form for your convenience. The last part reads LSN 1, the sector map. This section is a little slow. If you ever think the program has gotten lost, it is probably here working at counting available sectors. This part could be rewritten in assembly language, but I wanted to leave it in Basic09 to give a better idea of how it works. If you are interested in a C language version, I think you should have no trouble converting it. (Right now I have a sick disk drive. But as soon as it's fixed, I'll go to work on a C version.)

KANSAS CITY, HERE I COME

I just received a Basic Language for Coco OS-9 that I am excited about. It is KANSAS CITY BASIC (catchy name, eh?). Many of the old timers will remember the earlier basics, and may have used KC Basic. Many of them did not support floating point math. Only integer manipulations were permitted. KC BASIC is an Integer Basic.

This version of KC Basic is written by Steve Odneal. It runs on the RS Color Computer under OS-9. It handles basic syntax as well as some specialty items, like it can read the joy stick ports and it can pass a process to OS-9. (By the way, Steve points out that it should also run on standard OS-9 too.) It does lack a few things. As I said before, it can't do floating point math. It also can't handle arrays. So why am I so excited?

First, it resembles the original RS Color Computer's basic. If you've programmed the Coco you'll feel right at home with this. Another thing is that Steve supplies the Source Code with it. That's right! All you hackers will be able to see how it works. You can even add new commands if you feel so inclined, or the floating point math, etc.; if you add something, send it to Steve for incorporation into the package. Finally there is the price. Steve is asking $20 to cover the price of the Disk and Shipping. At that price you can't go wrong. If you're interested, you can write to:

Steve Odneal
8609 East 73 Terrace
Kansas City, MO 64133

Just tell him you saw it here in Basic OS-9. See ya next time!

```
PROCEDURE diskid

(* ********************************************)
(* THIS PROCEDURE WILL PRINT A DISK'S         *)
(* VITAL INFORMATION.  EITHER ACTUAL          *)
(* VALUES WILL BE PRINTED OR                  *)
(* AN INTERPRETATION WILL BE GIVEN.           *)
(* PROCEDURE SHOULD BE PACKED AND             *)
(* IN /D0/CMDS WITH RUN8.                     *)
(* CALLING SYNTAX:                            *)
(*      DISKID("DRIVE")                       *)
(* DRIVE ON MY SYSTEM IS /D0 OR /D1           *)
(* ********************************************)

(* DIMENSION VAR. AND ASSIGN CONSTANTS        *)
(*    "Drive" is  assed from the outside.     *)
(*    Variables in arrays are assigned        *)
(*    using READ and DATA statements.         *)

PARAM drive:STRING[10]
TYPE lsn0=b(31):BYTE; name:STRING[32]
DIM id:lsn0
DIM map:BYTE
DIM path,i,j,k,count,mapsize:INTEGER
DIM month(12):STRING[10]; attr(8):STRING[3]
DIM temp,lsn1,scount:REAL
drive:=TRIM$(drive)+"@"
BASE 0
FOR i=0 TO 11
  READ month(i)
NEXT i
DATA "January","February","March"
DATA "April","May","June","July"
DATA "August","September","October"
DATA "November","December"
FOR i=7 TO 0 STEP -1
  READ attr(i)
NEXT i
DATA "D","S","PE","PW","PR","E","W","R"

(* READ THE DISK'S INFORMATION INTO "ID"      *)
(*    path: path for "OPEN"                   *)
(*    drive: name of disk drive to read from  *)
(*    id: complex variable to hold disk info  *)
(*    GET is used to read into the complex    *)
(*    variable "id" the disk's information    *)
(*    from sector LSN 0.                      *)
OPEN #path,drive:READ
SEEK #path,0,0
GET #path,id
CLOSE #path

(* PRINT THE IDENTIFICATION FOR THE DISK      *)
(*    id.b: byte array of disk information    *)
(*    id.name: disk's name                    *)
(*    i, j, k, and temp: temporary variables  *)
(*    month: array of month names             *)
(*    attr: array of attributes               *)
(*    This section prints disk's information  *)
(*    in most cases PRINT USING is used       *)
(*    unless the printout is very simple or   *)
(*    or some effect is being created         *)
(*    that PRINT USING wouldn't help.         *)

PRINT "ID for disk in drive "; LEFT$(drive,3)
PRINT "Name: "; id.name
i:=id.b($10)-1
j:=id.b($1C)
k:=id.b($1A)
PRINT "Created on: "; month(i); " "; j; ", 19"; k
IF id.b($1E)<10 THEN
```

FINALLY !!!

Now you can run

TSC XBASIC Programs Compiled to Asmb. Lang., under **OS-9™, CoCo OS-9, or FLEX™ with**

☆ K-BASIC

K-BASIC under OS-9 and FLEX will now compile TSC BASIC, XBASIC, and XPC Source Code Files

K-BASIC now makes the multitude of TSC BASIC Software available for use under OS-9. Transfer your favorite BASIC Programs to OS-9, compile them, Assemble them, and BINGO -- usable, multi-precision, familiar Software is running under your favorite Operating System!

K-BASIC (OS-9 or FLEX), including the Assembler

$199.00

TELEX 558 414 PVT BTH

(615)842-4600

SOUTH EAST MEDIA

5900 Cassandra Smith Rd.
Hixson, TN 37343

for information
call (615) 842-4601

CoCo OS-9™ FLEX™

SOFTWARE

SCULPTOR

Microprocessor Developments Ltd.'s Commercial Application Generator Program provides a **FAST** Commercial Application Development tool unavailable to the OS-9 and UniFLEX User before. Develop any Commercial Application in 20% of the normal required time; gain easy updating or customizing. PLUS, the Application can also be run on MS-DOS and Unix machines! **Sculptor** handles input validation, complex calculations, and exception conditions as well as the normal collecting, displaying, reporting, and updating information in an orderly fashion. Key fields to 160 bytes; unlimited record size; file size should be held to 17 million records. Utilizes **ISAM** File Structure and B-tree Key files for rapid access. Input and Output communication with other programs and files plus a library of ISAM routines for use with C Programs. Run-time included w/ the Development package; a compiled Application only needs a Run-time License. Additional charge for Networked Units. Prices for **Development Package/Run-time.** Discounts available for purchases of **5 or more** Run-time Packages.

UniFLEX -- $1450.00/337.00
OS-9 Lvl II -- $995.00/200.00

Unix -- up to $2535/675 -- Call for info!
IBM PC & compats -- $635.00/170.00

☆ **BasicO9 Tour Guide**

by Dale Puckett -- An excellent Book on using OS-9. Oriented towards using the powerful **BasicO9 Programming Language**, it also contains a lot of good information on using OS-9 in general.

Normally $18.95
Special ---- NOW only $16.00

☆ **----- FLEX Software -----**

TSC "Flex Utilities"	was $75.00,	NOW only $65.00
TSC "Sort Merge"	was $75.00,	NOW only $65.00
TSC "6809 Basic"	was $75.00,	NOW only $65.00
TSC "Extended Basic"	was $100.00,	NOW only $90.00
TSC "DeBug"	was $75.00,	NOW only $65.00
TSC "FLEX Diagnostics"	was $75.00,	NOW only $65.00
TSC "Text Processing System"	was $75.00,	NOW only $65.00
TSC "68000 Cross Assembler"	was $250.00,	NOW only $199.95

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

VISA MasterCard

SOUTH EAST MEDIA
5900 Cassandra Smith Rd.
Hixson, TN 37343
CoCo OS-9™ FLEX™
info (615) 842-4601
SOFTWARE

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

*FLEX is a trademark of Technical Systems Consultants
**OS9 is a trademark of Microware

!!! Please Specify Your Operating System & Disk Size !!!

PROGRAMMING

ASSEMBLERS

ASTRUK09 from **Southeast Media** -- A "Structured Assembler for the 6809" which requires the **TSC Macro Assembler**. F, CCF - **$99.95**

Macro Assembler for **TSC** -- The FLEX **STANDARD** Assembler.
Special -- CCF **$35.00**; F **$50.00**

OSM Extended 6809 Macro Assembler from **Lloyd I/O**. -- Provides local labels, Motorola S-records, and Intel Hex records; XREF. Generate OS-9 Memory modules under FLEX. FLEX, CCF, OS-9 **$99.00**

Relocating Assembler w/Linking Loader from **TSC**. -- Use with many of the C and Pascal Compilers. F,CCF **$150.00**

MACE, by Graham Trott from **Windrush Micro Systems** -- Co-Resident Editor and Assembler; fast interactive A.L. Programming for small to medium-sized Programs. F,CCF - **$98.00**

TRUE CROSS ASSEMBLERS from **Computer Systems Consultants** -- Supports 1802/5, Z-80, 6800/1/2/3/8/11/HC11, 6804, 6805/HC05/146805, 6809/00/01, 6502 family, 8080/5, 8020/1/2/35/C35/39/40/48/C48/49/C49/50/8748/49, 8031/51/8751, and 68000 Systems. Assembler and Listing formats same as target CPU's format. Produces machine independent Motorola S-Text.
FLEX, CCF, OS-9, UniFLEX each - **$50.00**
any 3 - **$100.00**
the complete set w/ C Source (except the 68000 Source) - **$200.00**

XASM Cross Asseeblers for **FLEX** from **Compusense Ltd.** -- This set of 6800/1/2/3/5/8, 6301, 6502, 8080/5, and Z80 Cross Assemblers uses the familiar **TSC Macro Assembler** Command Line and Soruce Code format, Assembler options, etc., in providing code for the target CPU's. Complete set, FLEX only - **$150.00**

CRASM from **Lloyd I/O** -- 8-Bit Macro Cross Assembler with same features as OSM; cross-assemble to 6800/1/2/3/4/5/8/9/11, 6502, 1802, 8048 Sers, 80/85, Z-8, Z-80, TMS-7000 sers. Supports the target chip's standard mnemonics and addressing modes.
FLEX, CCF, OS-9 Full package -- **$399.00**

CRASM 16.32 from **Lloyd I/O** -- Cross Assembler for the **68000**.
FLEX, CCF, OS-9 **$249.00**

DISASSEMBLERS

SUPER SLEUTH from **Computer Systems Consultants** -- Interactive Disassembler; extremely POWERFUL! Disk File Binary/ASCII Examine/Change, Absolute or FULL Disassembly. XREF Generator, Label "Name Changer", and Files of "Standard Label Names" for different Operating Systems

Color Computer	SS-50 Bus (all w/ A.L. Source)
CCO (32K Req'd) Obj. Only $49.00	F, $99.00
CCF, Obj. Only $50.00	U, $100.00
CCF, w/Source $99.00	O, $101.00
CCO, Obj. Only $50.00.	

DYNAMITE + from **Computer Systems Center** -- Excellent standard "Batch Mode" Disassembler. Includes XREF Generator and "Standard Label" Files. Special OS-9 options w/ OS-9 Version.

CCF, Obj. Only $100.00	CCO, Obj. Only $ 59.95
F, " " $100.00	O, " " $150.00
	U, " " $300.00

PROGRAMMING LANGUAGES

PL/9 from **Windrush Micro Systems** -- By Graham Trott. A combination Editor/Compiler/Debugger. Direct source-to-object compilation delivering fast, compact, re-entrant, ROM-able, PIC. 8 & 16-bit integers & 6-digit Real numbers for all real-world problems. Direct control over ALL System resources, including interrupts. Comprehensive library support; simple Machine Code interface; step-by-step tracer for instant debugging. 500+ page Manual with tutorial guide. F, CCF - **$198.00**

WHIMSICAL from **Whimsical Developments** -- Now supports **Real Numbers**, "Structured Programming" WITHOUT losing the Speed and Control of Assembly Language! Single-pass Compiler features unified, user-defined I/O; produces ROMable Code; Procedures and Modules (including pre-compiled Modules); many "Types" up to 32 bit integers, **6-digit Real Numbers**, unlimited sized Arrays (vectors only); Interrupt handling; long Variable Names; variable initialization; Include directive; Conditional compiling; direct Code insertion; control of the Stack Pointer; etc. Run-Time subroutines inserted as called during compilation. Normally produces 10% less code than PL/9. F and CCF - **$195.00**

C Compiler from **Windrush Micro Systems** by James McCosh. Full C for FLEX except bit-fields, including an Assembler. Requires the TSC Relocating Assembler if user desires to implement his own Libraries. F and CCF - **$295.00**

C Compiler from **Introl** -- Full C except Doubles and Bit Fields, streamlined for the 6809. Reliable Compiler; FAST, efficient Code. More UNIX Compatible than most.
F, CCF, and O - **$375.00** U - **$425.00**

PASCAL Compiler from **Lucidata** -- ISO Based P-Code Compiler. Designed especially for Microcomputer Systems. Allows linkage to Assembler Code for maximum flexibility.
F and CCF 5" - **$190.00** F 8" - **$205.00**

PASCAL Compiler from **OmegaSoft** -- For the **PROFESSIONAL**; ISO Based, Native Code Compiler. Primarily for Real-Time and Process Control applications. Powerful; Flexible. Requires a "Motorola Compatible" Relocating Asmb. and Linking Loader.
F and CCF - **$425.00** One Year Maint. - **$100.00**

K-BASIC from **LLOYD I/O** -- A "Native Code" **BASIC Compiler** which is now **Fully TSC XBASIC** compatible. The compiler compiles to Assembly Language Source Code. A NEW, streamlined, Assembler is now included allowing the assembly of LARGE Compiled K-BASIC Programs. Conditional assembly reduces Run-time package.
FLEX, CCF, OS-9 Compiler with Assembler - **$199.00**

CRUNCH COBOL from **Compusense Ltd.** -- Supports large subset of ANSII Level 1 COBOL with many of the useful Level 2 features. Full FLEX File Structures, including Random Files and the ability to process Keyed Files. Segment and link large programs at runtime, or implemented as a set of overlays. The System requires 56K and CAN be run with a single Disk System.
FLEX, CCF; Normally $199.00
Special Introductory Price (while in effect) -- **$99.95**

FORTH from **Stearns Electronics** -- A CoCo FORTH Programming Language. Tailored to the CoCo! Supplied on Tape, transferable to disk. Written in FAST ML. Many CoCo functions (Graphics, Sound, etc.). Includes an Editor, Trace, etc. Provides CPU Carry Flag accessibility, Fast Task Multiplexing, Clean Interrupt Handling, etc. for the "Pro". Excellent "Learning" tool!
Color Computer ONLY - **$58.95**

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

SOUTH EAST MEDIA
5900 Cassandra Smith Rd.
Hixson, TN 37343
Info (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware

!!! Please Specify Your Operating System & Disk Size !!!

GENERAL

SOFTWARE DEVELOPMENT

BasicO9 XRef from Southeast Media -- This **BasicO9 Cross Reference** Utility is a BasicO9 Program which will produce a "pretty printed" listing with each line numbered, followed by a complete cross referenced listing of all variables, external procedures, and line numbers called. Also includes a **Program List Utility** which outputs a fast "pretty printed" listing with line numbers. Requires BasicO9 or RunB.
O & CCO obj. only -- **$39.95**; w/ Source - **$79.95**

Lucidata PASCAL UTILITIES (Requires LUCIDATA Pascal ver 3)

XREF -- produce a Cross Reference Listing of any text; oriented to Pascal Source. F and CCF - $25.00

INCLUDE -- Include other Files in a Source Text, including Binary; unlimited nesting capabilities. F and CCF - $25.00

PROFILER -- provides an Indented, Numbered, "Structogram" of a Pascal Source Text File; view the overall structure of large programs, program integrity, etc. Supplied in Pascal Source Code; requires compilation. F and CCF - $25.00

DUB from Southeast Media -- A **UniFLEX** "basic" De-Compiler. Re-Create a Source Listing from UniFLEX **Compiled basic** Programs. Works w/ ALL versions of 6809 UniFLEX basic. U - **$219.95**

FULL SCREEN FORMS DISPLAY from Computer Systems Consultants -- **TSC Extended BASIC** program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.
F and CCF - **$50.00**, U - **$75.00**

DISK UTILITIES

OS-9 YDisk from Southeast Media -- For **Level I** only. Use the **Extended Memory** capability of your SWTPC or Gimix CPU card (or similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increments up to 960K. Some Assembly Required.
-- **Level I ONLY** -- OS-9 obj. only - **$79.95**; w/ Source - **$149.95**

O-F from Southeast Media -- Written in **BASICO9 (with** Source), includes: **REFORMAT**, a BASICO9 Program that reformats a chosen amount of an OS-9 disk to FLEX Format so it can be used normally by FLEX; and **FLEX**, a BASICO9 Program that does the actual **read** or **write** function to the special **O-F Transfer Disk**; user-friendly menu driven. Read the FLEX Directory, Delete FLEX Files, Copy both directions, etc. FLEX users use the special disk just like any other FLEX disk. O - **$79.95**

COPYMULT from Southeast Media -- Copy LARGE Disks to several **smaller disks**. FLEX utilities allow the backup of **ANY size disk** to any SMALLER size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply inserting diskettes as requested by **COPYMULT**. No fooling with directory deletions, etc. **COPYMULT.CMD** understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes **BACKUP.CMD** to download any size "random" type file; **RESTORE.CMD** to restructure copied "random" files for copying, or recopying back to the host system; and **FREELINK.CMD** as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation.
Completely documented Assembly Language Source files included.
ALL 4 Programs (FLEX, 8" or 5") **$99.50**

COPYCAT from Lucidata -- Pascal **NOT** required. Allows reading TSC Mini-FLEX, SSB DOS68, and Digital Research CP/M Disks while operating under FLEX 1.0, FLEX 2.0, or FLEX 9.0 with 6800 or 6809 Systems. COPYCAT will not perform miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Also includes some Utilities to help out. Programs supplied in **Modular Source Code** (Assembly Language) to help solve unusual problems.
F and CCF 5" - **$50.00** F 8" - **$65.00**

FLEX DISK UTILITIES from Computer Systems Consultants -- Eight (8) different Assembly Language (w/ Source Code) FLEX Utilities for every FLEX Users Toolbox: **Copy** a File with **CRC Errors**; **Test** Disk for errors; **Compare** two Disks; a fast **Disk Backup** Program; **Edit Disk Sectors**; **Linearize Free-Chain** on the Disk; print **Disk Identification**; and **Sort** and **Replace** the Disk Directory (in sorted order). -- **PLUS** -- Ten XBASIC Programs including: A **BASIC Resequencer** with EXTRAs over "RENUM" like check for missing label definitions, processes Disk to Disk instead of in Memory, etc. Other programs **Compare**, **Merge**, or **Generate Updates** between two BASIC Programs, check BASIC **Sequence Numbers**, compare **two unsequenced** files, and 5 Programs for establishing a **Master Directory** of several Disks, and sorting, selecting, updating, and printing paginated listings of these files. A BASIC **Cross-Reference** Program, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved Words in **TSC BASIC**, **XBASIC**, and **PRECOMPILER** BASIC Programs.
ALL Utilities include **Source** (either BASIC or A.L. Source Code).
F and CCF - **$50.00**

COMMUNICATIONS

CMODEM Telecommunications Program from Computer Systems Consultants, Inc. -- Menu-Driven; supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "Modem7" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".
FLEX, CCF, OS-9, UniFLEX; with Complete Source - **$100.00**
without Source - **$50.00**

XDATA from Southeast Media -- A **COMMUNICATION** Package for the **UniFLEX** Operating System. Use with CP/M, Main Frames, other UniFLEX Systems, etc. Verifies Transmission using checksum or CRC; Re-Transmits bad blocks, etc. U - **$299.99**

GAME

RAPIER - 6809 Chess Program from Southeast Media -- Requires FLEX and Displays on **Any** Type Terminal. Features: Four levels of play. Swap side. Point scoring system. Two display boards. Change skill level. Solve Checkmate problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. This is one of the **strongest** CHESS programs running on any microcomputer, estimated USCF Rating 1600+ (better than most 'club' players at higher levels). F and CCF - **$79.95**

VISA

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

*FLEX is a trademark of Technical Systems Consultants
*OS9 is a trademark of Microware

!!! Please Specify Your Operating System & Disk Size !!!

TELEX 558 414 PVT STN

(615)842-4600

SOUTH EAST MEDIA

5900 Cassandra Smith Rd.
Hixson, TN 37343
for information
call (615) 842-4601

CoCo OS-9™ FLEX™

SOFTWARE

BUSINESS

WORD PROCESSING

SCREDITOR III from Windrush Micro Systems -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!

6800 or 6809 FLEX or SSB DOS, OS-9 - **$175.00**

STYLO-GRAPH from Great Plains Computer Co. -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/STAR-DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.
NEW PRICES —> CCF and CCO - **$99.95**, F or O - **$179.95**, U - **$299.95**

STYLO-SPELL from Great Plains Computer Co. -- Fast Computer Dictionary. Complements Stylograph.
NEW PRICES —> CCF and CCO - **$69.95**, F or O - **$99.95**, U - **$149.95**

STYLO-MERGE from Great Plains Computer Co. -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.
NEW PRICES —> CCF and CCO - **$59.95**, F or O - **$79.95**, U - **$129.95**

STYLO-PAK --- Graph + Spell + Merge **Package Deal!!!**
F or O - **$329.95**, U - **$549.95**

JUST from Southeast Media -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the Display. Use the FPRINT.CMD supplied for producing multiple copies of the "Formatted" Text on the Printer INCLUDING IMBEDDED PRINTER COMMANDS (very useful at other times also, and worth the price of the program by itself). "User Configurable" for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with **ANY** Editor. Supplied with "Structured Source" (Windrush PL/9); easy to see the flow of the program.

F and CCF - **$49.95**

SPELL8 "Computer Dictionary" from Southeast Media -- **OVER 120,000 words!** Look up a word from within your Editor or Word Processor (with the SPH.CMD Utility which operates in the FLEX UCS). Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELL8 first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELL8 also allows the use of **Small Dist Storage** systems.

F and CCF - **$129.95**

DATA BASE - ACCOUNTING

XDMS from Westchester Applied Business Systems -- Powerful DBMS; M.L. program will work on a **single sided 5" disk**, yet is F-A-S-T. Supports Relational, Sequential, Hierarchical, and Random Access File Structures; has Virtual Memory capabilities for **Giant Data Bases**. **XDMS Level I** provides an "entry level" System for defining a Data Base, entering and changing the Data, and producing Reports. **XDMS Level II** adds the POWERFUL "**GENERATE**" facility with an English Language Command Structure for manipulating the Data to create new File Structures, Sort, Select, Calculate, etc. **XDMS Level III** adds special "**Utilities**" which provide additional ease in setting up a Data Base, such as copying old data into new Data Structures, changing System Parameters, etc.

XDMS System **Manual** - **$24.95**
XDMS Lvl I - F & CCF - **$129.95**
XDMS Lvl II - F & CCF - **$199.95**
XDMS Lvl III - F & CCF - **$269.95**

ACCOUNTING PACKAGES -- **Great Plains Computer Co.** and **Universal Data Research, Inc.** both have Data Base and Business Packages written in TSC XBASIC for FLEX, CoCo FLEX, and UniFLEX.
Call 800-338-6800 for more information

MISCELLANEOUS

TABULA RASA SPREADSHEET from Computer Systems Consultants -- TABULA RASA is similar to DESKTOP/PLAN; provides use of tabular computation schemes used for analysis of business, sales, and economic conditions. Menu-driven; extensive report-generation capabilities. Requires **TSC's Extended BASIC.**

F and CCF - **$100.00**, U - **$200.00**

DYNACALC from Computer Systems Center -- Electronic Spread Sheet for the 6809.

F and **SPECIAL** CCF - **$200.00**, U - **$395.00**

FULL SCREEN INVENTORY/MRP from Computer Systems Consultants -- Use the Full Screen Inventory System/Materials Requirement Planning for maintaining inventories. Keeps item field file in alphabetical order for easier inquiry. Locate and/or print records matching partial or complete item, description, vendor, or attributes; find backorder or below stock levels. Print-outs in item or vendor order. MRP capability for the maintenance and analysis of Hierarchical assemblies of items in the inventory file. Requires **TSC's Extended BASIC.**

F and CCF - **$100.00**, U - **$150.00**

FULL SCREEN MAILING LIST from Computer Systems Consultants -- The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or Labels, etc. Requires **TSC's Extended BASIC.**

F and CCF - **$100.00**, U - **$110.00**

DIET-TRAC Forecaster from Southeast Media -- An XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate. Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and calorie plan is determined.

F - **$99.95**, U - **$89.95**

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

SOUTH EAST MEDIA
5900 Cassandra Smith Rd
Hixson TN 37343
info (615) 842-4601
CoCo OS-9* FLEX* SOFTWARE

*FLEX is a trademark of Technical Systems Consultants
*OS9 is a trademark of Microware

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

!!! Please Specify Your Operating System & Disk Size !!!

```
  PRINT "At: "; id.b($10); ":0"; id.b($1E); ":00"
ELSE
  PRINT "At: "; id.b($10); ":"; id.b($1E); ":00"
ENDIF
i:=id.b($0B)*256+id.b($0C)
PRINT USING "S9,I6,S2","Owner ID: ",i,"  ";
i:=id.b($0E)*256+id.b($0F)
PRINT USING "S9,I6","Disk ID: ",i
temp:=id.b($00)*65536.+id.b($01)*256+id.b($02)
PRINT USING "S9,I6","Sectors: ",temp;
PRINT USING "S12,I3","     Tracks: ",id.b($03)
temp:=256*(id.b($06)*256+id.b($07))
lsn1:=temp
PRINT USING "S14,I5","Bytes/Sector: ",temp
PRINT "Attributes: ";
j:=id.b($0D)
FOR i=7 TO 0 STEP -1
  IF LAND(j,2**i)<>0 THEN
    PRINT " "; attr(i);
  ENDIF
NEXT i
PRINT
PRINT "Format: ";
IF LAND(id.b($10),1)=0 THEN
  PRINT "SS ";
ELSE
  PRINT "DS ";
ENDIF
IF LAND(id.b($10),2)=0 THEN
  PRINT "SD ";
ELSE
  PRINT "DD ";
ENDIF
IF LAND(id.b($10),4)=0 THEN
  PRINT "48 TPI ";
ELSE

  PRINT "96 TPI ";
ENDIF
PRINT
i:=id.b($15)
j:=id.b($16)
k:=id.b($17)
IF i=0 AND j=0 AND k=0 THEN
  PRINT "Disk is not bootable"
ELSE
  PRINT "Disk is bootable"
ENDIF

(* DETERMINE NUMBER OF FREE SECTORS          *)
(*   count: the number of bytes read         *
(*   mapsize: the size of the sector map     *
(*   scount: number of free sectors counted  *
(*   lsn1: location of LSN 1                 *
(*   This area reads the number of bits that *
(*   are turned off in the bytes in the      *
(*   maP area.                               *

count:=0
mapsize:=id.b($04)*256+id.b($05)
scount:=0
OPEN #path,drive:READ
SEEK #path,lsn1
WHILE count<mapsize DO
  GET #path,map
  count:=count+1
  FOR i=0 TO 7
    IF LAND(map,2**i)=0 THEN
      scount:=scount+1
    ENDIF
  NEXT i
ENDWHILE
CLOSE #path
PRINT USING "S19,I6","Available Sectors: ",sco

END
```

CoCo User Notes

MODEM COMMUNICATIONS, Part One;
or,
A Modem in Every Port

by Carl Mann

It almost makes sense, sometimes. After all, a computer can transmit information faster than all but the most nimble of us can think it - and with (allegedly) fewer errors in the process. So why not use an inexpensive, powerful micro (like CoCo, just for example) as a means of transferring lots of information between distant points over, say something common and easy to use - like telephone lines? So the reasoning goes - and thus were the various hobbies that comprise modem communications born.

Just for the sake of our new readers, (welcome, friends!) let's back up a wee bit at this point. Us old-timers might tend to take the fundamentals for granted sometimes, so let's have a little background for starters, OK? If you will, Maestro...

(At this point, the lights dim. An eerie blue high-tech glow is as much felt as seen, and the emcee's voice takes on a compelling quality of resonance thanks to studio enhancement.)

MODEM: As with many elements of the primitive pseudo-language known to many as "Compubabble", this word is an acronym. (The actual equipment referred to may well be an anachronism, depending on its age and the market for which it was designed.) It stands for "MOdulator/DEModulator". The reference to modulation (and its undoing) in turn refers to the way in which the device works. A modem, like most "black boxes", has in "in" end and an "out" end. Such a device accepts digital information at its input and superimposes it onto a special tone signal on its output. The tone (with data superimposed) may then be transmitted over much greater distances than the raw data alone could ever be. The tone is called the "carrier", because it "carries" the data. The data is just called "data".

Modems are generally designed to operate at a specific rate of data transfer. This transfer rate is called the "BAUD" rate. Don't bother to look it up in the RS Microcomputer Dictionary. (It's a very good book for technical detail, but pretty short on background and history.) Webster's Collegiate defines it well. The "BAUD" rate is named after the French inventor Baudot, who presumably pioneered some early form of electromechanical or electrical communication about the early part of the century. A data transfer rate of one BAUD is equal to transmitting information at the rate of one electrical vibration, or "bit", per second.

One bit per second is far too slow to be useful. The minimum rate for CoCo communications is generally 300 BAUD. Let's see... that's one "start" bit, eight data bits, and a "stop" bit. Ten bits in all, and only one character (the letter "A",

say) transmitted. Call an average word five letters long, and you've got roughly 60 words per second flying down Ma Bell's Pipeline to the Stars. That's much faster than anybody I know can read - or type, for that matter.

Modems come in all sizes and price ranges, depending on the design, features, and time of year. Fifty bucks is more or less the low end, and several hundred the high. The cheap ones require that you do everything - dialing, mode selection, etcetera - all by hand. But they are easy to learn to use, and generally effective. (The Shack sells its 28-1175 Modem 1B for as little as $40.00 sometimes.) More expensive units will dial the 'phone (including Sprint or other discount long-distance service) codes, listen for a pickup at the other end, and keep on hanging up and trying again until something (or someONE, which is a definite hazard) picks up the 'phone at the other end. Top-of-the-line modems will handle multiple BAUD rates, do everything else I mentioned before, PLUS transmit any number of pre-programmable messages designed to enable you to reach into the cookie jar at the other end, grab what you want, and get out with a minimum of wasted "connect-time". (That, of course, is what Ma Bell duns you for whenever you make any long-distance call - modem or not.) Some public access information services (commonly called "BBSs", for Bulletin Board Service(s)) and all commercial information services slap an additional charge of five to forty or more dollars per hour onto the cost of hooking up. On the other hand, many privately-owned and club BBSs allow at least limited access without charge. ("Goodies - really useful information or programs) may cost extra. Shop around. Many computer clubs offer lists of active BBSs from their own BBS databases. ("Database" is compubabble for "a pile of information organized in some semblance of order".)

The modem is the "hardware" (that is, the copper, fiberglass, epoxy, and silicon) of the job. The computer program ("software", the babble goes...) which enables all this wondrous action to happen before your eyes is another story. May I make a suggestion? Get the least expensive modem that will get you on line - and the best, most sophisticated terminal package available. You do NOT need a disk system to have a working communications package. I have seen an intelligent modem make a monkey out of three grown men for the better part of two weeks. But excellent software is a definite must. Look for such things as:

1: A wide range of available baud rates - especially on the high end. Nobody uses 110 baud anymore, but 2400 baud is becoming increasingly popular. It's easier to relearn how to use a new modem than it is to relearn a new communications software package.

2: The ability to control what information is actually saved for later use, as opposed to what is simply scrolled past your eyes and off the video screen back into the Great Void. This ability should include two aspects: local and remote control. Remote control allows the host computer to tell your CoCo, "WAKE UP AND LISTEN!" before it sends a free program, a pretty picture, or the irreplaceable contents of your Electronic Mailbox Account across the Gulf of Time and Space.

3: Control over EVERY aspect of the "communications protocol". This includes the number of start, stop, data, and parity bits used to communicate and test the data for errors. Having this capability enables you to communicate with everything from Commodore VIC-20s all the way up through the mega-computers that run our country's libraries and schools. Not to mention the odd minicomputer that might be encountered on the way to the grocery store. (You, not the mini, I presume...)

4: Last, but not least: a menu-oriented display of the above communication parameters that allows YOU, the OPERATOR, to SEE what the actual values ACTUALLY ARE at any point in time BEFORE making any adjustments (as may happen in mid-session). There's nothing worse than needing to adjust, say, the "word length" (that's actually "bits per character", friends) in midstream, but not remembering what it presently is in the heat of the moment - and not being able to find out unless one passes it by with the <ENTER> key first. The only way to deal with such a program is to remember the necessary information, then re-enter the menu and make the necessary changes on the second pass. Not nice. Insist on visible parameters when you go shopping.

Once all the appropriate hardware and software are assembled and running it's

just a matter of connect-time and practice. Once you learn where all the buttons are and when to push them, CoCo-as-communicator may become as much a part of the family as anyone else.

The nice thing about such a microcomputer as CoCo is that it is a "literally anything" machine. Today CoCo processes words. Tonight CoCo picks up satellite weather pictures. Tomorrow it's in control of a monster mainframe computer halfway across the continent. The next day, it's back to remembering recipes and 'phone numbers. Where else can you go to tell a box full of plastic and silicon, "YOU ARE A (insert function here)" and expect instant obedience? Not bad for a lousy two hundred bucks or so.

Until next time...

The DRAGON Engine

UNIX-LIKE TOOLS ON SINGLE USER/SINGLE TASK COMPUTERS

By
Tom Gilchrist
Brad Taylor

You may not have enough money to have a computer running UNIX(*), but you can have some of the power of UNIX running on your personal computer. Some of the features of UNIX tools can be simulated in a single-user, single-task environment. A set of functions have been assembled which will help you put together UNIX-like tools. We call these functions the "Dragon" engine. Two UNIX-like tools are presented which use these functions and will serve as examples of how other tools can be written.

The example programs are written in INTROL-C for the FLEX(*) operating system. FLEX is for the 6809 processor and is used in a number of general purpose computers like the Radio Shack Color computer and computers by Smoke Signal, just to name two manufacturers. The techniques used can be adapted to operating systems like MS/DOS or CP/M.

THE UNIX SHELL

The "shell" in a UNIX system can be thought of as a powerful "command line processor". In other words, when you type in a command, the shell interprets the command and executes the desired action. The shell is a program that always runs when you are in the command mode of UNIX.

A good example of some of the features of the UNIX shell can be illustrated by some examples. There is a command, or "tool" in UNIX terminology, called "ls". The function of this tool is to list the contents of a directory in much the same way as the "dir" or "cat" command found, in many disk operating systems, list the files on a disk. As a default, the "ls" tool lists all the files in your current working directory in alphabetical order. This is a welcome feature when you are looking through a long listing of disk files.

Another feature of the shell is it's ability to do "wild card" searches on file names. With this feature you can display files that contain certain desired character patterns in their names. The two more common matching symbols are the "?" and the "*". When the "?" is used in a pattern mask, it is used to match any single character. The "*" tells the shell to match anything. There is a third basic mask construct which is "[x-y]" which matches a single character in the range of x to y. A few examples will demonstrate this feature.

```
ls  test*
ls  hello?.txt
ls  [a-g]ver.c
```

The first example will match any file name starting with the string "test" or the word "test" by its self. The second example will list all the files which have 6 characters and start with "hello" and contain any one character before the ".txt" extension. "helloa.txt" would be found with this mask, but "hello.txt" and "helloer.txt" would not. The third example will match any file that starts with any one of the characters a through g followed by "ver.c". The file "bver.c" would be listed but "wver.c" would

not. By combining these three basic masks you can construct very useful and powerful masks.

COMMAND LINE ARGUMENTS

There is another powerful feature of the UNIX shell which deals with passing arguments to a tool or program. Let's consider the UNIX tool called "grep". This tool will search a file for the occurrence of a given set of characters. If we were looking for the word "print" in the file "text.txt", we would type the following command line.

```
grep  print  text.txt
```

Many operating systems have the tool "find" that will do this task. However, UNIX gives us the power to use a wild card mask in the file name to tell the tool to search through any number of files.

```
grep  print  *.txt
```

In this example the system would find all the occurrences of the string "print" in any file that has the extension of ".txt" and report them. The UNIX tool "grep" will not only list the files in which it found the match, but will also list them in alphabetical order.

If you think about this last example, there are at least three ways an operating system could handle this task. Let's say there are three files in the directory that will match the mask "*.txt". The names of these three files are "a.txt", "ab.txt", and "abc.txt".

One method might be to have the shell run the tool "grep" three times. Each time the shell would call the tool with the correct file name inserted in place of the mask. While this sounds reasonable at first, it is really not a good idea for a number of reasons. For instance, it would be very difficult to have the tool give the total number of instances the string was found in all three files as a statistic at the end of the listing.

A second method would have the shell pass the program the list of file names which matched the mask. It would be like typing in the following line.

```
grep  print  a.txt  ab.txt  abc.txt
```

It would then be up to the program to open each file, processes it, then close it, one at a time. This is basically the way the UNIX shell works.

A third way would be to pass the command line to the tool and let it do all the work of finding the file names which match the mask. The tool would also have to alphabetize the list and process each file as in the second example. If you don't have a shell like UNIX on your computer, the Dragon engine, using this method, will accomplish the task.

THE GENERAL DRAGON ENGINE

Whether you are writing tools under UNIX or a small DOS on a personal computer, you will need to design some sort of "engine" to process command line arguments. The "Dragon" engine is designed to do some of the tasks done by the UNIX shell. Writing tools in 'C' allows the code and technique to be very portable between different operating systems.

The general case for the engine should be designed to deal with command line options (or dash options) as well as file names. While the Dragon engine does not have all the features you might need for all tools, it is sufficient for most tasks. Each program uses the file "dragon.c" for support routines. Below is a structured english representation of the first level of the Dragon engine.

```
Get command line arguments.
Check for dash options and set flags.
Get the Mask (if applicable).
Open the disk directory.
Read A Directory Entry.
    Process the file name against the
mask.
    Store matching file names in
alphabetical order.
Close Directory.
Get each file name in order stored.
    Pass Each file name to the
"action" section of the tool.
Exit the tool (back to DOS).
```

The listings of two tools using the Dragon engine start on page XXXX. The first listing is for a tool called "grep". The "grep" tool will search for given character strings within text files. The second tool is called "ls". As described earlier, the "ls" tool gives a listing of files in a directory.

These two tools are portable to other operating systems except for the Dragon engine support routines in "dragon.c". The only major changes in "dragon.c" are the routines called "diropen()", "dirread()", and "dirclose()". These routines will have to be customized for different operating systems.

You should also be aware that the engine can be either case sensitive or not in respect to the Mask. The dash option "-c" is used in "ls" to signal the engine to make the mask case sensitive. Just about all disk operating systems will allow you to enter a disk file name in either upper or lower case when at the console. However, how DOS saves the file names in the disk directory can be another matter. In FLEX, the "-c" option tells the engine to make the mask case insensitive. Of course, you can hard wire the case flag and take out the "-c" option if you want.

DRAGON FOR FLEX

The FLEX operating system is a single user operating system which is essentially single tasking. Getting file names from disk directories is handled by system calls to FLEX via their File Management System or FMS. The general technique used to access the directory is that of treating the directory as a file.

FLEX is designed on the concept of a default drive for both system (where programs normally reside) and work (where data files reside). Because of this fact, we can assume that if a drive is not explicitly defined, we can use the drive number found in the structure element "FLEX_DATA.work_drive" supplied by the compiler. This work is done in a routine call "diropen()". This routine checks the Mask to see if a drive number has been defined. The general case for a FLEX file name is...

drive_number.file_name.extension

The drive_number is a single digit between 0 and 3. The file_name can contain up to 8 characters and the extension can be up to 3 characters.

The routine also gets the name of the disk and the volume number of the disk. These two items are defined by the user when a disk is formatted. Finally, the routine prints the name and volume number to "stdout" (the standard output which is normally the CRT).

FLEX requires only that the drive number be put into a File Control Block or FCB. Then the FCB structure is loaded with an action flag instructing FLEX to open the directory block on the disk using the "_fms()" call. The address of this structure is then passed back to the calling routine.

The "dirread()" is passed this address (ptr) and is used to get directory entries each time the "_fms() routine is called with a "get directory entry" flag set. There is logic to detect entries in the directory which are not valid, as well as the EOF (end of directory entries). The drive number, file name, and extension are appended together to make the complete file name stored in "filename".

The "dirclose()" routine is not needed in FLEX and is included only as a stub for systems requiring it.

FILE MASK

The routine "getfnam()" in Dragon is used to compare a file name with a mask given by the user in the command line. This routine calls "mskcmp()" which is a replacement for the UNIX wild card mask system in the shell. This is a general routine which can be used any time you want to use a wildcard mask inside a program. The function header on the "mskcmp()" function gives a definition of the wild card syntax available. While the syntax is not as powerful as the syntax in the "sh" and "csh" UNIX shells, it incorporates almost all the features. The routine also gives a good example of the ability of 'C' to use recursive calls (the "mskcmp()" routine calls itself).

SORTING

The Dragon uses the "sortin()" function to sort file names in ascending alphabetical order ignoring case. The routines "sortin()" and "node()" build a tree using the structure "SYMBOL". The tree structure used comes from an algorithm by Knuth in the book "THE ART OF COMPUTER PROGRAMMING, VOL 1". The tree allocates memory using the 'C' standard library function "sbrk()". If memory can not be allocated, an error message is given.

The "traverse()" routine is used to get the file names off the tree. When a file name is retrieved, it is passed to the "action()" function where the actual work of the tool is done. This routine exits once

all the file names have been retrieved.

Because of the general nature of this sort, it can be used in other programs when you want to sort variable length strings.

THE "GREP" TOOL

This tool is used to find a pattern in a line of a file which matches the pattern given in the command line when calling the tool. It will search all file names matching the file name mask for occurrences of the pattern. The files are searched one at a time in alphabetical order. When a match is found, both the line number and the contents of the line are displayed. The calling line is given below.

```
grep  pattern  file_name_mask
```

The pattern is the character string you wish to search (no spaces are allowed in the pattern in FLEX) and the pattern is never case sensitive. The file_name_mask is a legal FLEX file name which can include the Dragon wild card characters.

ACTION()

The "action()" portion of the "grep" tool uses the routine "fstring()" to open a file and search for the search string pattern. If the file is opened successfully, each line of up to 256 characters, is read, copied into a temporary buffer (line1), and then searched using the function "findstr()". INTROL-C does not include "findstr()" in its standard library, so it is included in the program. Some compilers include this function. If it is included in the compiler you are using, and it matches the definition, you can use it instead of the one given.

If a line contains a match, the original line buffer (line) is printed along with its line number. When all the lines have been processed, the file is closed and the function returns back to "traverse()" via "action()". When all files have been processed off the sorted tree, the tool returns to FLEX.

THE "LS" TOOL

The "ls" tool is used to find directory entries which match the given file name mask. The entries found are listed in alphabetical order. The syntax for the command line is given below.

```
ls  [-c(C)l]  file_name_mask
```

The "-c" dash option will make the file_name_mask insensitive to case. The "-l" option will list the files vertically instead of the usual horizontal format. The dash options are checked in the function "options()".

```
ls  * (or just ls)
ls  -c  *.txt
ls  -l  1.???.*
```

The first example will list all directory entries in the current working directory. The second example will list all entries in the current working directory ending with ".txt". The "-c" means that both "TXT" and "txt" extensions will be found. The third example will find all files with exactly three characters in their names with any extension. This last example will list the entries found in vertical format.

ACTION()

In the "ls" tool, if no arguments are given on the command line, we assume that all files on the work drive directory are to be listed. If this is the case, an "*" is copied into the mask.

As in the "grep" tool, the directory is searched for file names which match the mask given. Each file name that matches the mask is added to the sort tree. When the directory search is done, the tree is traversed and each file name is passed to the "action()" routine.

The action routine in "ls" simply prints out the file names either in horizontal columns or vertically depending on the command line arguments. One added feature is the printing of the total number of files found before returning to FLEX.

CONCLUSION

For those who program in 'C', the sort routines can be added to your library if you don't have "sort()" in your standard library. The "mskcmp()" routine ("mskcmp()", "onecmp()", and "maskup()") can be used for an "in tool" wild card routine. On the other hand, reading directories "in tool" with wild cards in UNIX is really quite easy. A good article on utilizing UNIX "in tool"

for wild card inquiries can be found in the November 1982 issue of Dr. Dobbs called "Expanded Wildcards Under UNIX" by Anthony Skjellum. If you are interested in adding complex arguments to your tools, the August 1982 issue of Dr. Dobbs contains a good library for command line parsing called "ARGUM" also by Anthony Skjellum.

By using 'C', and the Dragon engine routines, many different tools are possible. You can probably see how you could write a "delete" tool to delete all files that match a given mask. Other tools could include "list", "copy", and "wc" (word count) just to name a few. The technique for writing even more tools is given in the book "SOFTWARE TOOLS" by Kernighan and Plauger. There are at least two versions of the book I know of, one for PASCAL and one for RATFOR.

(*) Footnotes:

UNIX is a trademark of Bell Laboratories

FLEX is a trademark of Technical Systems Consultants

INTROL-C is a trademark of Introl Corporation

Tom Gilchrist
1450 N. Clarence #108
Wichita, KS 67203

Brad Taylor
611 E. Helbert
Mulvane, KS 67110

The source code in this article is available on the C.Dragon Software Exchange System at (316) 943-9716 1200/300 baud 24 hours.

** Also 6809 systems from SWTPC - GIMIX - AAA Chicago - WINDRUSH - PT-69 - UniBoard - ST-2900 - MicroKey - Hazelwood and many others. Any system running FLEX(c) and Introl C(c) can process these programs. Which I find very useful. Thanks fellows, and we need more of this type article. C is coming strong and also we still have many left doing it in assembler, FORTH, PL-9, Whimiscal, Pascal, and yes, even BASIC. So no matter what language you program in, we can sure use your input.

With all the Cobols we have sold, where are some in Cobol? One nice thing about 'C'. If you send in an article with a source listing in C, then everyone (FLEX and OS-9) can compile and run it. NICE! Lets have more utility type artices, especially with a good explanation of how they work.

DMW

```
/*****************************************
*
*    GREP for FLEX
*    by
*
*    Brad Taylor &
*    Tom Gilchrist
*
*    For FLEX on 6809
*    Using INTROL-C
*
*****************************************/
#include "stdio.h"
#include "DRAGON.C"

/* Globals */

char match[128];    /*  hold match string  */

/*
**** Main
*/

main(argc,argv)

int  argc;
char **argv;

{

    FCB  *ptr;
    char filename[32];
    int  count;

    if(argc < 3)
        usage();

    strcpy(match,argv[1]);    /*  copy argv  */
    strcpy(mask,argv[2]);

/** Open Directory for read **/

    ptr = diropen(mask);

/** Read Directory entry **/

    while(dirread(ptr,filename) != ERROR)
        {
            if(getfnam(filename))
                {
                    ++count;
                    sortin(filename);
                } /* if */
        } /* while */

/** End of Directory **/

    dirclose(ptr);

    if(!count) /* no files found */
        {
            fprintf(stderr,"\nFind: No files match '%s'\n",mask);
            usage();
        } /* if */

    traverse(treep);    /*  traverse sort tree for files */

} /* grep */

/**
*** Error Routine
**/

usage()

{

    fprintf(stderr,"Usage: find string file_name\n");
    exit(0);

} /* usage */

/*
```

```
*** action routine
*/

action(s)

char *s;

{

    fstring(s);

} /* action */


/*
** fstring find string within file(s)
*/

fstring(file)

char *file;

{

   FILE *fd;
   int  flag,i;
   int  j,k;
   char line[257];
   char line1[257];
   char file1[30];

/* put on the D_drive number */

    sprintf(file1,"%d.%s",D_drive,file);

    if((fd=fopen(file1,"r")) == ERROR)
        {
            fprintf(stderr,"\nError Opening file %s\n",file);
            return;
        } /* if */

    i=flag=0;

    while(fgets(line,256,fd))
        {
            ++i;
            strcpy(line1,line);
            makup(line1);
            makup(match);
            if(findstr(1,line1,match))
                {
                    if(!flag)
                        {
                            printf("\n----------------\n");
                            printf("File: %s:\n",file1);
                            flag=1;
                        } /* if */

                    printf("%5d - %s",i,line);
                } /* if */

        } /* while */

    fclose(fd);

} /* fstring */


/*
*** findstr(pos,string,pattern)
*
*   This function searches for the pattern in
*   the string and returns the position.
*
*   Both pos and the returned position are 1 based.
*/

findstr(pos,s,s2)

int  pos;
char *s,*s2;
       char *pat1,*pat2,*s1;
       int  len1,len2,cnt;

       len1 = strlen(s +(pos -1));
       len2 = strlen(s2);
       if(len1 < len2)
           return(0);

       s1 = s +(pos -1);

       cnt = 0;
       len1 -= len2;
       while(len1--)
           {
               pat1 = s1++;
               pat2 = s2;
               while(*pat1 == *pat2)
                   {
                       pat1++;
                       pat2++;
                   } /* while */

               if(!*pat2)   /*  end of string found */
                   return(cnt +pos);

               cnt++;
           } /* while */

    return(0);

} /* findstr */


/*****************************************
*
*    ls for FLEX
*    by
*
*    Brad Taylor &
*    Tom Gilchrist
*
*    For FLEX on 6809
*    Using INTROL C
*    2/24/84
*
*****************************************/
#include "stdio.h"
#include "DRAGON.C"


/* Globals */

char flag1;
char tally;
int  nof;


/*
**** Main
*/

main(argc,argv)

int  argc;
char **argv;

{

    FCB  *ptr,*diropen();
    char filename[30];
    int  count,i;
    char name[20];
    char ext[5];

    count = flag1 = nof = 0;

    i=1;
    options(argc,argv,&i);    /*  compute flag options */

    if(!argv[i])
        strcpy(mask,"*");
    else
        strcpy(mask,argv[i]);    /*  put mask in global string */

    if(sensitive)
        makup(mask);    /* ignore case */

    ptr=diropen(mask);        /*  attempt to open directory */

    /** Read Directories entries **/

    while(dirread(ptr,filename) != ERROR)
        {
            if(getfnam(filename))
                {
                    ++count;
                    sortin(filename);
                } /* if */
        } /* while */

    if(!count) /* no files found */
        {
            printf("\nls: No files match '%s'\n",mask);
            usage();
        } /* if */

    dirclose(ptr);      /*  close directory */
```

```
    traverse(treep);
    printf("\nNumber of files found = %d",nof);

}


/*
*** check global options
*/

options(n,p,i)

int  o;
char **p;
int  *i;      /*  index of current argument to parse */

{
    char *cp;

    tally = sensitive = 0;

    while(--n && *(cp = p[*i]) == '-')
        {
            if(!cp[1])
                usage();
            else
                {
                    ++*i;
                    while(*++cp)
                        switch(tolower(*cp))
                            {

                                case 'l':
                                    tally=1;
                                    break;

                                case 'c':
                                    sensitive=1;
                                    break;

                                default:
                                    usage();

                            } /* switch */
                } /* else */
        } /* while */

} /* options */


/**
 ** Error Routine
**/

usage()

{

    puts("Usage: ls [-[l][c]] [filename]\n");
    exit(0);

} /* usage */


/*
*** action routine
*/

action(s)

char *s;
{

    nof++;
    if(tally && flag1++ != 4)
        {
            if(strlen(s) <8)
                printf("%s\t\t",s);
            else
                printf("%s\t",s);
        } /* if */
    else
        {
            printf("%s\n",s);
            flag1 = 0;
        } /* else */

} /* action */
```

```
/*********************************************
*                                            *
*     dragon.c                               *
*                                            *
*     General Purpose find/sort              *
*     engine routines.                       *
*     For TSC 6809 FLEX                      *
*     Using INTROL-C                         *
*                                            *
*     By                                     *
*     Brad Taylor                            *
*     Tom Gilchrist                          *
*                                            *
*********************************************/

#include "flex.h"  /* Part of INTROL C */
#define MEMERR -1

/* Globals */

typedef struct fcbc FCB;
typedef struct nd
                {
                 char *sym;
                 char *left;
                 char *right;
                 int  symsiz;
                } SYMBOL;
SYMBOL *treep = NULL   ;     /* symbol pointer */

char mask[128];
char sensitive = 0;     /*  case sensitivity flag */
int  D_drive;           /*  Directory drive number */


/*
***  Return File Name
*/

getfnam(file)

char *file;

{

    if(!makcmp(file,mask))
       *file='\0';
    return(*file);

} /* getfnam */

/*
**  Sort Symbol Onto Tree
*/

sortin(f)

char *f;

{
    SYMBOL **parent,*current,*node();

    parent=&treep;
    while(current=*parent)
       if(cplcap(f,current->sym)<0)
          parent=&(current->left);
       else
          parent=&(current->right);

    return(*parent=node(f));

} /* sortin */


/*
*** Construct A Tree Node
*/

SYMBOL *node(key)

char *key;

{

    char   *cptr;
    SYMBOL *loc;

    if((loc = (SYMBOL *) sbrk(sizeof(SYMBOL))) == MEMERR)
       giveup("Symbol Bucket");

    loc->left = loc->right = NULL;
    if((cptr = sbrk(strlen(key)+1)) == MEMERR)
       giveup("Symbol Definition");
```

```
    loc->sym = cptr;
    strcpy(cptr,key);
    return(loc);

} /* node */


/*
*** Giveup The Ghost
*/

giveup(why)

char *why;

{

    fprintf(stderr,"Not enough memory for %s\n",why);
    exit(1);

} /* giveup */

/*
*** Traverse Tree
*/

traverse(active)

SYMBOL *active;

{
    SYMBOL *sub;

    if((sub = active->left))
       traverse(sub);

    action(active->sym);      /* perform action */
    if((sub = active->right))
       traverse(sub);

} /* traverse */


/*
***  Special String Compare Routine
***
***  This routine treats upper and lower
***  case identically.
*/

spclcmp(s,t)

char *s,*t;

{

    int c1,c2;

    while((c1 = tolower(*s++)) == (c2 = tolower(*t++)) && c1);
    return(c1-c2);

} /* spclcmp */


/*
*** Convert String To Upper Case
*/

makup(s)

char *s;

{

    while(*s = tolower(*s))
       ++s;

} /* makup */


/*
*** Compare Against A Mask
*
*    This function compares a given string against a mask and returns
*    a 1 for 'compares' or a 0 for 'does not compare'.
*
*    A mask is a concatenation of the following elements:
*
*    c          literal character
*    ?          any character match except endstring null
*    [..]       character class (all of these characters)
*    [^..]      negated character class (all but these characters)
*    ^c         negated character (all but this character)
*    *          some (match zero or more occurences)
*
*    A character class consists of zero or more of the following
*    surrounded by [ and ]:
*
*    c1-c2      range of ASCII characters
*    c1-c2,..c1-c3 multiple ranges
*
*
*/

makcmp(string,m)

char *string,*m;

{

    int k;
    char *sp,*sav,string2[128];

    strcpy(sp=string2,string);
       if(sensitive)
          makup(sp);

    while(*m)
       {
           if(*m =='*')
               {
                   sav = sp;
                   if(!*++m)
                      return(1);

                   while(*sp && !makcmp(sp,m))
                      ++sp;

                   if(*sp)
                      continue;

                   sp = sav;
               } /* if */
           else
              if(!(k=onecmp(*sp,m)))
                 return(0);
              else
                 m += k;

           if(*sp)
              ++sp;

       } /* while */

    return(!*sp);

} /* makcmp */

/*
*** Compare Only One Character (for makcmp)
*/

onecmp(s,m)

char s,*m;

{

    char c,setfind,setflag;
    char *mp;

    if((c = *(mp=m)) =='?' && s);  /* okay as is */
    else if(c == '[')
       {
           setfind = setflag = 0;
           if(*++mp =='^')
               {
                   setflag=1;
                   ++mp;
               } /* if */

           for(;(c = *mp) && c!=']';++mp)
               if(*mp=='-' && s>= *(mp-1) &&
               s<= *(mp+1) && *(mp-1) <= *(mp+1))
                   {
                       /* skip to trailing ']' */
                       while((c= *(mp+1)) && c!=']')
                          ++mp;

                       setfind=1;
                   } /* if */

               if(setfind==setflag)
                   return(0);
               else
                   return(mp-m+1);
       } /* else */

    else if(c == '^' && *(mp+1) != s)
        return(2);
```

```
    else if(c != s)
        return(0);

    return(1);

} /* onecmp */


/*
*** Open Disk Directory
*/

FCB *diropen(name)

char *name;
{

    FCB *ptr;
    int  i;
    char dname[16];     /*  disk file name */
    int  volnum;        /*  volume #       */

    D_drive = FLEX_DATA.work_drive;

    if(strlen(mask) == 1 && isdigit(*mask))
        strcat(mask,":");

    if(mask[1] == ':') /* a file drive has been specified */
       {
          if(isdigit(*mask))
              {
                  D_drive = *mask-'0';
                  strcpy(mask,mask + 2);
              } /* if */
       } /* if */

    /* Check for driver error */

    if(D_drive <0 || D_drive > 3)
        usage();

    /* Set fcb for directory */

    if((ptr=(FCB *)sbrk(sizeof(FCB))) == MEMERR)
        giveup("directory open.");

    ptr->f.drive = D_drive;

    /* Get SIR */

    ptr->f.function = IN_OPEN;
    if(_fms(ptr,0) == ERROR)
        usage();

    ptr->f.function = GET_INF;
    if(_fms(ptr,0) == ERROR)
        giveup("directory open.");

    for(i = 0;i<11;++i)
        dname[i] = ptr->f.filename[i];

    volnum=(ptr->f.attributes)*256 + ptr->f.rful;
    dname[11] = '\0';

    /* Open Directory for read */

    ptr->f.function = DR_OPEN;
    if(_fms(ptr,0) == ERROR)
        usage();

    /* print header */

    printf("Disk: %s #%d\n\n",dname,volnum);

    return(ptr);

} /* diropen */

/*
*** Read A Directory Entry
*/

dirread(ptr,filename)

FCB  *ptr;
char *filename;

{

    int i,j;

    ptr->f.function = GET_INF;
    while(_fms(ptr,0) != ERROR)
        {
            for(i = 0;i < 8;++i)
                filename[i] = ptr->f.filename[i];

            filename[8] = '\0';
            if(*filename && !((*filename)&128))
                {
                    strcat(filename,".");
                    j = strlen(filename);
                    for( i = 0; i < 3; i++)
                       filename[j+i] = ptr->f.extension[i];

                    filename[j + 3] = '\0';
                    return(1);
                } /* if */
        } /* while */

    return(ERROR);

} /* dirread */

/*
*** Directory Close
*/

dirclose(ptr)

FCB *ptr;

{

/* Not Used In Flex */

}
```

Bit Bucket

Where Is The CoCo Going?

Editor's Note: We, CPI (68 Micro Journal) will continue to support the CoCo. However, we are hearing sad things often concerning the state-of-affairs for Tandy (Radio Shack) computers, which is the bulwark of their returns.

Seems like the information I released some time back concerning their 'old faithful line' of computers (Models 1,2,3 and 4) is becoming more a fact each day. They are soon to lose another member. The Model 4 will bite the dust as some of the others have. Not that those series of computers ever attracted much attention in our sphere of communication. However, it is to be taken with more than a grain of salt, when it comes to the CoCo.

Tandy has a nasty habit of running off and leaving sick and dying models. Most should have been put to sleep long before anyway, but it sure makes it tough when you own one of those little suckers, and go down to the local store only to be told, "We don't know nothing about that one anymore." If it wasn't for outside support the problem would be many-fold worse.

Fortunately the CoCo has enjoyed more than it's share of outside support. For instance, never did ANY Tandy computer have so popular a group of dedicated magazines. Of which we were the very first. But not most popular (we told it like it is or was). Despite its low cost and amazing power Tandy looked upon the CoCo as the 'taken in orphan'. Simply because its 'profit per unit' was so much less than some of the other systems - now GONE! And soon it also will be gone, that is as we know it today. There may be another "Color Computer Model ?" on the way, but I am willing to bet heavily that it will not be the low priced item, it now is. Not that it could not be, but then the pricing will be what the market will bear. Remember the cost of the original 4K CoCo?

Today I was reading an industry news letter I receive each couple of weeks (you think 68 Micro Journals subscriptions are high? - Boy!) and it seems that Tandy has lost much more market share than I had though, or been led to believe. In Europe alone (where some others are showing substantial gains, Tandy is down from about 9% market share to 3% or less. Also reported was that they had taken massive write-offs of something like $18 million, here in the USA. Beginning to sound like the old TI story?

All of which saddens me much. And I guess I really can't blame them much. If I were Tandy, and had the reputation and preceptions afforded by their customers and the general public, I guess I would also become a "ME TOO - BIG BLUE"! But what happens when BB makes a sharp turn?

The CoCo could have been the beginning of a nice and probably profitable line of Tandy computers. Folks really liked the little jewel. Could be yet, plenty of support still out there. But fast receding. Not BB cloned, just a nice machine for the average Joe Blow who does not need a lot of bells and whistles, but does need support. But even now a lot of those who were zinging along last year on CoCo sales are now ill and ailing. Business in the CoCo world is down, way down. It is a shame, it could have been a "Hallmark" of a system and although never approach the number of BB, it could have been a profitable thing for Tandy, I think they blew it!

We will probably hang in there longer than Tandy does with the CoCo - but because of you, who spent your hard earned bucks for a fine little system, that is soon to get lost. I just hope the something new(?) can pull the load.

DMW

- - -

Dear Larry:

Please find enclosed the paper on the KSR 43 which we discussed over the phone a few weeks ago. I sincerely hope that you can use it, and that it will be of some service to another 43 owner.

I shall be looking for its publication in the future.

Steve Conley

*

***

W2FOG
One Whitney Road
Latham NY
12110
USA

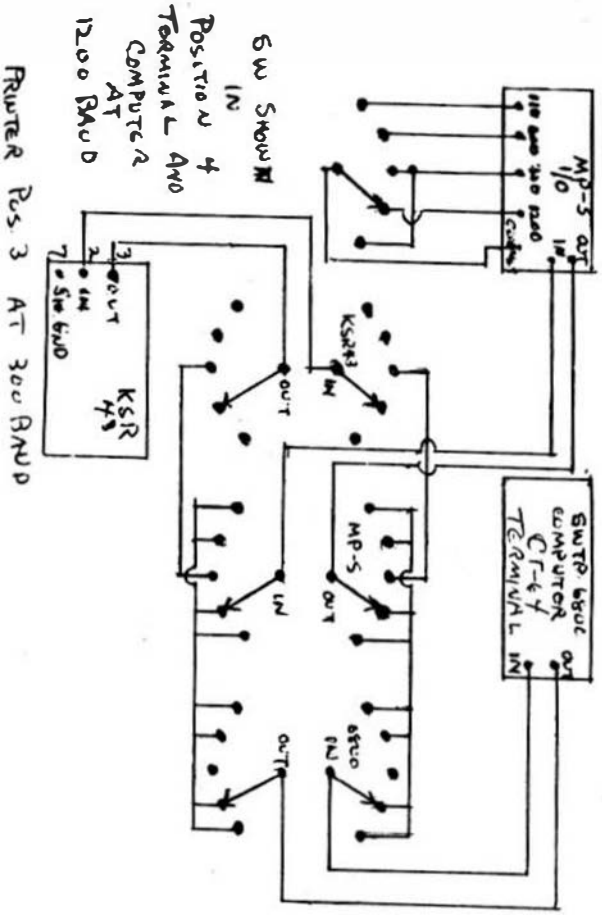
*

Having had the good fortune to aquire a TT KSR43, I then was faced with the task of interfacing to the MPS I/O board on my SWTPC 6800 computer. I was previously using a TT ASR33 on the TT current loop circuit of the MPS in tandem with my CT-64 Video Terminal. since I was now faced with two RS232 ports to feed and only one I/O port on the MPS, I wired a seven section five position rotary switch to choose one of four Baud rates and either the CT-64 terminal or the KSR43. The switch was wired to operate the terminal at 1200 Baud and the KSR43 at 300 Baud.

The TAU circuit board supplied with the KSR43 needs two jumpers installed for operation. One jumper is wired to pins 4 and 5, and the second to pins 6, 8, and 20. I found it much simpler to wire the jumpers directly on the printed wiring of the board. These are the only modifications necessary to the KSR43. Standard RS232 connector is used to interface to the KSR43. A standard male RS232 connector is used to interface the female RS232 input on the KSR43, using terminals 2, 3, nd 7.

See figure 1 for details of the wiring on the switch mentioned earlier. It is possible to simplify the circuit, if you do not need as many Baud rates as shown.

Good luck. If problems call me at 518-785-5089 or write above address.



I would like to comment on two articles in the May 1985 issue of 68MJ. First, in Bud Pass's "C" User Notes, there is an ambiguity in the example on page 15 that reads

```
if(n < 10) a[n++] = n;
```

In this example, the value of the subscript is the current value of "n" (before incrementing it), but the value of "n" to the right of the equal sign may be either the old or the new value. Which value will be used depends on the compiler and is not specified in the definition of the C language. For example if "n" has the value 5, then the assignment is to a[5], but the value assigned could be either 5 or 6, depending on whether the "n" to the right of the equal sign is evaluated before or after incrementing it.

This situation, using a slightly different statement, is discussed by

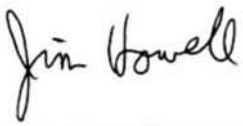
Kernighan and Ritchie in "The C Programming Language" at the end of Chapter 2 on page 50.

Regarding Peter Dibble's call for games for OS-9 (in OS-9 User Notes), I have written a program for my SS-50 OS-9 system that plays cribbage (a card game). Since I wrote it in C, I can (and have) ported it to an MS-DOS machine and this version is currently in the hands of a software company for evaluation. Sharing the OS-9 version of cribbage is partly dependent on what happens (if anything) at this or some other company, but also awaits working out another problem.

Making cribbage run on "all" OS-9 systems is a problem due to the various terminals that are being used. The program uses cursor positioning and "clear to end of screen". The character sequences for these vary considerably among different kinds of terminals. Even screen sizes (number of lines and columns) can vary somewhat. I am currently using a 16 by 64 display with my OS-9 system, while most terminals are 24 by 80. Building in one particular screen size and one particular set of special character sequences is not hard. It is much harder to make the program configurable for ANY terminal. I would be interested in any suggestions along these lines. A few possibilities come to mind. (1) A separate configuration program could be written which asks the user about the terminal being used and writes a data file with appropriate information. (2) An ASCII file containing the configuration data could be set up by the user (following supplied instructions) and the program could read this file when starting. (3) The C source could be supplied and the user could be required to edit the source for the terminal-dependent functions and then compile (and link) the result. A few "standard" configurations could be supplied. (4) A set of terminal-dependent subroutines could be written as a separate module which would be callable by cribbage (and other programs as well). This module would be written by the user for his/her particular terminal, though a few sets of these subroutines could be supplied for "standard" terminals. (5) The configuration file of an existing program, such as Screditor III or Dynacalc, could be used, which of course requires the user to own at least one of several commercial programs. For example, I use Screditor III for editing. Cribbage could be written to use Screditor III's console configuration file to determine which sequences to use for cursor positioning (etc.) and how big the screen is.

It would be desirable to be able to use the configuration information with other programs besides just cribbage. All of the above except number 3 could be used this way. Number 4, however, does not take care of the screen size differences, but otherwise is rather intriguing.

In summary, one of the difficulties in writing games (and other programs) for systems that use terminals is the problem of accomodating the differences in "control sequences" among various terminals. Is this a problem that could be "solved once" and then used by everyone? I am interested in how others have approached this problem. Other 68MJ readers may be interested also.



Jim Howell

5472 Playa Del Rey
San Jose, CA 95123

MRDY and the "Synchronizer Problem"

Malcolm D. Muir
Computer Excellence Inc.
P.O. Box 8442
Coral Springs, Fl. 33075
(305) 752-8321

Recently I discovered the cause of a rather nasty intermittent problem in a system in our office. Since it appears that other users may have the same problem I hereby pass on what was found.

When it appeared in our environment the main symptoms were picking up extra characters with the system sitting in an input loop and other I/O related errors. In our case these extra characters would appear about once every 10 minutes.

When the problem was finally trapped on a logic analyzer it became obvious that this was an example of the "synchronizer Problem".

Over the last ten years there have been numerous papers published about the "Synchronizer Problem" or the "Metastable Flip Flop" problem. In essence the theory is that if a flip flop is given a runt signal to change state it may partially change state. This tends to show up as one of three behaviors at the output as a function of the design of the flip flop. Either the flip flop may do an imitation of a one shot producing a pulse output rather than a full state transition, or it may oscillate producing a burst of pulses (or sine waves) or it may sit with one or both outputs in the transition region. With a clocked flip flop a runt signal may arise from violating the

setup and hold requirements of data with respect to the clock. According to theory ALL flip flops will produce this behavior to a greater or lesser extent.

The SS-50 bus problem with synchronizers occurs with the MRDY signal. Not MRDY is typically generated by a one shot and is used to slow down bus cycles for I/O etc. The 6809 samples MRDY with an internal free running clock and if the sample occurs during the transition of MRDY a metastable state may occur. What was seen here was that when the metastable state occurred the first 1/4 of the instruction following the slow cycle had disappeared and that instruction did not operate correctly. The window of vulnerability is quite small and small changes in loading of MRDY may significantly change the error probability. As a function of exact timing, data and address patterns, one shot jitter, power supply regulation, noise, etc. the error probability may be expected to vary widely.

NOTE: The 6809 spec sheet talks about synchronizing MRDY to the internal clock for certain early mask sets. That requirement relates to the leading edge of the wait signal. This phenomenon relates to the trailing edge of MRDY and is present in all 6809 processors. Motorola does not publish any spec information on the trailing edge timing requirements.

The symptoms mentioned here were as a result of metastable behavior caused by the slow I/O one shot on the backplane. The same phenomenon, in other MRDY one shots (DMA disc controllers, etc) would show different symptoms.

This phenomenon is not easy to detect (It was never visible on an oscilloscope) but it is easy to fix. Replace the timing resistor on each MRDY one shot with a pot (choose a pot with a value greater than the resistor which is being replaced, up to twice the value). The pot should be installed with the wiper and one end to one lead and the other end to the other lead. Adjust that pot for a time interval on MRDY low which is about half way between the intervals which cause E clock to change delay. If mounting a pot is impractical due to mechanical constraints temporarily wire in a pot and adjust it, then without disturbing the adjustment remove the pot and measure the resistance. Install a fixed resistor with a resistance as close as possible to the measured resistance.

If you switch CPU cards it may be necessary to readjust the pots.

Floating Point Library for the 6809:

We would like to inform the readers of '68' Micro Journal about our new FAST/09 Floating Point Package for the 6809 chip. Its fast execution times, number of implemented functions and ease of use make it particularly useful for industrial and scientific applications:

- Code size 2Kbytes, PIC, re-entrant
- 32bit Real format, 7 significant digits
- Trig functions: SIN,COS,TAN,ASIN,ACOS,ATAN
- Logarithmic functions: LOG,LN,PWR,e^x,10^x
- I/O conversions: ASCII/Float, Float/ASCII
- 16bit Integer/Float,Float/Integer conv's
- All parameters passed on 6809 user stack
- Optimized instruction timing using Chebyshev polynomials
- Integrated FORTH interface

The library is available under FLEX, OS-9 (Sinch) or ROM. Please contact

Amrein System Ingenieure
Gerberstrasse 1
CH-4410 Liestal, Switzerland
Phone (061)91 3045

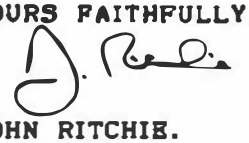
for further informations.

45 COOK ROAD,
NEWLANDS,
CAPE TOWN, 7700
SOUTH AFRICA.

DEAR DON,

RE: LOG.CMD 68MJ MAY 85.
ANOTHER BUG HAS COME TO MY ATTENTION. THE VALIDITY OF THE FILESPEC IS NOT CHECKED. THIS CAN BE EASILY CORRECTED BY INSERTING 'LBCS BAD' AFTER THE LINE 'JSR GETFIL'.

YOURS FAITHFULLY,



JOHN RITCHIE.

Dear Mr. Williams:

In your April 1985 issue I saw an announcement of a hard disk system from Wellwritten Enterprises. I had been looking for a hard disk to install on my OS-9 system and wrote Wellwritten to ask for a brochure describing their system. What I got in return was not a brochure but a two-page, single spaced description of their system and the purchase options. It provides much more information than the announcement did.

I have enclosed a copy of the letter because the information in it may be interesting to readers who are considering moving up to a hard disk. (The copy of the letter is not as good as I would have liked because it was typed with a brown ribbon on cream colored paper.)

Since I received the letter, I have spoken to Tom Weaver of Welwritten and he

told me that they are currently completing an interrupt-driven version of the SS-30 host adapter card. He says, it runs even faster than the current programmed input/output model.

Keep up the good work.

Sincerely,

Ken Drexler

Kenneth Drexler

Dear Sirs,

We are writing you to let you know of a new bulletin board being operated in the Forth Worth - Dallas (Tex) area for the benefit of TRS-80 Color Computer users.

TBBS Fort Worth is available 24 hours daily at (817) 232-2087, at either 300 or 1200 baud.

Unique features of TBBS Fort Worth include bugs and fixes for Tandy's Color Computer software line, numerous technical reference files, and a large database of public domain software, including quite a few OS-9/BASIC09 files.

No fees are charged, and first time callers have full access to the board with the exception of leaving messages in public view. Registration requires only a name, address and telephone number, and higher access is usually granted within 12 hours.

We would appreciate anything you could do to help spread the word.

Thank you,

R. Wayne Day - aysop
TBBS Fort Worth

MICRONICS

3383 LYNN AVENUE,
ABBOTSFORD,
BRITISH COLUMBIA,
CANADA, V2S 1E2

Microcomputers - Hardware and Software
OMIX® Sales, Service and Support

68' Micro Journal
5900 Cassandra Smith Rd.,
Hixson, TN 37343

Dear Don,

From time to time I use TSC's EDITOR to edit XBASIC programs, particularly wherer global changes become necessary. As you know, TSC's EOL character defaults to a colon, which makes it impossible to edit this VERY common character in XBASIC lines without the hassle of redefining the EOL char in the EDITOR. Similarly with the DCC (Don't Care Char). So I decided to patch EDIT to initialise itself to characters which are rarely found in TEXT editing, and while I was about it I also set up the TAB char to a backslash (/). EOL was set to a tilde (~) and DCC to a grave accent (`). The changes to accomplish this are :

1. Using the system monitor, set address-range 0000 - 1AFF to 00, then return to FLEX and execute GET 0.EDT.CMD
2. Back to system monitor, and using the Memory Examine and Change function locate the HEX string B6 CC02. In my version of EDIT this occurs at address 0353, so yours should be close.
3. Commencing at address 0353 enter the following code :

```
0353  86  7E  (EOL separator equals ~)
0355  A7  C821
0358  B6  CC00
035B  A7  C810
035E  B6  CC07
0361  C6  60 (DCC equal `)
0363  ED  C81E
0366  B6  CC01
0369  A7  C820
036C  86  5C (TAB equal /)
```

4. Return to FLEX and execute SAVE 0.EDIT.CMD 0000 1A07 0000.

Of course, if you have a DISKEDIT utility command, it is easier and quicker to edit in the changes directly to the sector concerned.

Another problem with TSC's EDITOR is that no calls can be made to other FLEX utilities. The following patch solves this problem.

1. As above, GET the EDIT command into memory, and locate it Directive Table - quite near the start of memory. We are going to remove the directive OVERLAY to make room for a new directive FLEXCMD. Anyway, who would use "OVERLAY" when the shortened form "O" is available? Unfortunately, the directive MUST be in alphabetical order, so we have to bubble-down a block of memory to wipe out OVERLAY and make room in the Fa for FLEXCMD.
2. This is done by moving the block commencing with FLUSH and ending with N 00 xxxx (where xxxx will be some address) forward by 10 address locations. In my version of EDIT, this means moving the block 0088 - 00EC to a new position commencing at 0092.
3. Now, commencing at location 0088 I entered FLEXCMD 00 19A3 (19A3 being one location beyond the last address occupied by EDIT). This can easily be found beforehand with MAP command. Ignore the address of the final solitary byte, which will typically be displayed as 1A07-1A07, as this is the start of EDIT's buffer. The word FLEXCMD should, of course, be entered in ASCII as 46 4C 45 58 43 4D 44. That's the most difficult part done!!
4. Finally, we append the following code commencing (in my case) at 19A3 :

19A3 + 15 bytes

```
8E 19 B2 BD CD 1E BD C0 1B BD CD 4B 7E 00 03
0D 0A 46 4C 45 58 20 43 6F 6D 6D 61 6E 64 20
2E 2E 2E 04
```

and then do step 4 of the first example above.

You are now all set, but keep in mind that no FLEX command which would overwrite either EDIT or its buffer should ever be called from EDIT. I would NOT recommend making this last change with a DISKEDIT command, unless you are VERY familiar with the structure of binary files on disk, as the number of bytes in the final block of code has now been changed, and you will also have to re-enter the code for the start-of-buffer and the link address, which will have been overwritten by the changes above.

Sincerely,

B. Jaso
President

TYETRONICS

A Division of Stylo-Software, Inc.
TYPESETTING • TELECOMMUNICATIONS—TYPESETTING • TYPESETTER INTERFACING
P.O. Box 818 482 C Street, Idaho Falls, ID 83402

From: Gary J. Duarte, Tyetronics
Date: January 31, 1985
Re: For Immediate Release:

Tyetronics and Stylo-Software, Inc. team up with a high level word processing system to drive the major typesetting devices. Gary J. Duarte of Tyetronics has announced he has reached an arrangement with Stylo-Software, Inc. of Idaho to develop a powerful word processor that will fully drive the major digitized typesetting devices. Stylo-Software, Inc. owner of Stylo-Graph word processing will be integrating total typesetter coding into this already successful program, this and the typesetting program will be called Stylo-Type. The program is being moved from 6809 technology to the MC68000 chip to run on the Apple Macintosh system. This program is not just an interface to the typesetters,

but a micro font end system. The Macintosh system eventually will be able to generate "What you see is what you get" from its screen to the dot matrix printer and then ultimately to a typesetting output device.

The Stylo-Type program is estimated to cost about $1,500. A total system configuration for the Stylo-Type program, communications, Macintosh, and the Apple Imagewriter printer for proofing purposes is expected to run about $4,995 through $7,995 depending on the options.

Stylo-Software, Inc. is a certified developer for Apple computer systems. Stylo-Type version 1.0 is expected to be a basic interface program without the display capability. This version will be available in February of '85. The first output device driven will the the Mergenthaler CRTronic family of which there are some 5,000 in the field. The Macintosh Stylo-Type system will be a powerful input station for the CRTronic owners. It will also provide a word processing oriented typesetting input device for advertising agencies, publishers, author, printers etc. and once their inputting is completed, they can phone download for the typesetting output.

Tyetronics is also setting up typesetting throughput facilities, the first one in Idaho Falls, ID and the second in Reno Nevada which will service the Macintosh front end systems sold. However, the Stylo-Type system will be capable of typesetting its files to any CRTronic system world wide equipped with Mergenthaler LCI communications program and a special configuration for the CRTronic.

Tyetronics and Stylo-Software, Inc. will continue to develop specialized programs for the graphics industry. One specialty program nearly completed is a semi-automated book manuscript/castoff paging and page cost estimating system. Stylo-Software, Inc. also is moving an entire group of programs to the Macintosh system which include accounting, database management, terminal communications, mail lists, spelling, etc. that will enhance a full line of STYLO integrated software.

For more information you may contact Gary Duarte or of Stylo-Software, Inc.(208)529-3210.

GIMIX INC. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX 910-221-4055

PRESS RELEASE
68020 DEVELOPMENT SYSTEMS TO BE INTRODUCED AT THE 1985 NCC

BOOTH #3726

The GMX 68020 is a multi-user, multi-tasking, demand-page, virtual memory system. The GMX 68020 has provisions for an optional 68881 floating point co-processor.

The base system hardware includes 512K Bytes of high speed RAM, 3 intelligent serial ports, a 39MB hard disk, and a 1 MB floppy disk. The power supply of this expandable system uses a constant-voltage, ferro-resonant transformer and has sufficient reserve capacity to support additional users and multiple high performance, high capacity, hard disks. Intelligent Serial I/O Boards significantly reduce system overhead by handling routine I/O functions, freeing the host CPU for running user programs. This improves overall system performance and allows user terminals to be run at up to 19.2K baud.

The UniFLEX VM Operating System, is modeled after UNIX System V and is written in assembler code optimized for rapid execution times and kernel compactness. As a result, UniFLEX runs several times faster than UNIX systems, handles more users more effectively, and leaves more disk space for the end user. It is UNIX compatible at the C source level. A GMX version of Motorola's 68020 Bug is also included. Languages available include Ada, C, Cobol, Fortran, Basic, Assembler, Franz Lisp, Prolog.

Export models are available.

For further information contact:
Richard Don at (312) 927-5510

GIMIX, Inc., a Chicago based microcomputer company established in 1975, has produced state of the art microcomputer systems based on Motorola 6800 and 6809 microprocessors. GIMIX systems are in use in Industry, Hospitals, Universities, Research Organizations, and by Software Developers. GIMIX was awarded the prestigious President's "E" Certificate for Exports in 1984.

We invite you to visit the GIMIX booth #3726 at NCC to see the GMX 68020 Supermicro at work.

DEMAND-PAGED VIRTUAL-MEMORY OPERATING SYSTEM
FOR GMX 68020 DEVELOPMENT SYSTEM

Chapel Hill -- Technical Systems Consultants, Inc., has released a demand-paged virtual-memory version of its UniFLEX™ Operating System which is specifically optimized for the GMX 68020 Development System manufactured by GIMIX, Inc.

This release is the culmination of years of research and development into software designed exclusively for the Motoola 68xx and 68xxx family of microprocessors. Although the UniFLEX system offers most of the features found in UNIX™ systems, its performance is dramatically better due to an assembly language implementation and specific optimization for the Motorola 68xxx family.

The UniFLEX system is the only currently available demand-paged virtual-memory operating environment which is specifically designed for the 68020 and which may be adapted to systems ranging from ROM based applications, to single-user, multi-tasking workstations, to full multi-user, multi-tasking, time-sharing systems. A typical kernel, without any device drivers will reside in approximately 28K of memory. The kernel with I/O drivers, variable storage, and all static tables will reside in approximately 50K to 60K of memory. Approximately 1.7 Megabytes of disk storage is required for the operating system, utilities, assembler, loader, C compiler, C library, and help files. An additional one to three megabytes is required for swap space.

Technical Systems Consultants, Inc., was formed in 1976 and specializes in operating systems, languages, and utilities for Motorola 68xx and 68xxx family of microprocessors. The multi-user, multi-tasking, UniFLEX Operating System has been running on 6809 based hardware since September, 1980, and on 68000 based hardware since October, 1982. Further information may be obtained from Technical Sytems Consultants, Inc., 111 Providence Road, Chapel Hill, NC 27514 919-493-1451 or TWX 510-920-0540 TSC CPEL.

Contact: Don Sinkiewicz
Director of Marketing
Technical Systems Consultants, Inc.
111 Providence Road
Chapel Hill, NC 27514

(919) 493-1451

MOTOROLA INC.

Microprocessor Products Division
3501 Ed Bluestein Blvd
Austin, Texas 78721

MOTOROLA ANNOUNCES NEW
MICROPROCESSOR DEVELOPMENT SYSTEM

Phoenix, April 23, 1985... Motorola announces the MDS-300-a low-cost microprocessor/microcontroller Hardware Development Station that provides real-time hardware/software emulation support for the MC68HC11 Microcomputer and the MC6809/MC6809E and M6801/03 Microprocessor Families. The MDS-300 is a stand-alone development system that requires only the target object file from the host system. Almost any terminal with a standard RS-232C port can be used with MDS-300.

By selecting the proper Emulator Module, users can interface the HDS-300 to a variety of Motorola Microprocessors. The HDS-300 consists of a control station with a built-in 5 1/4" disk drive for program execution to the target code, user macros, and terminal configuration data. It also features a real-time signal-tracing bus state monitor, a built-in assembler/disassembler for the target processor, external synchronization I/O for opertion with other development tools (or multiple processor opertion), built-in real-tim trace with disassemble, 32Kb of high-speed RAM (expandable to 256Kb), and bus transfer rates of 4MHz, with no wait states.

The HDS-300 allows software and hardware to be developed and tested at an early stage of the development process, permitting hardware configuration and software code changes to be made while still at the prototyping stage. Original algorithms and I/O parameters can be tested before the final software code is written. After this stage is completed, a source program is prepared by using the host's editing facilities. Three suitable development hosts include the VME/10, the EXORmacs multiuser station, and the IBM PC (and compatibles).

For development support on the MC68HC11, the MC68HC11 Cross C Compiler provides an efficient, high-level language for product development. For hosted environments, Source Level Debug (SLD) allows applications witten in "C" to be compiled and/or assembled then linked into Common Object File Format. The code is then downloaded into the HDS-30 station's emulation memory. Comprehensive diagnostics, including a power self-test, are included with HDS-300 system.

READER CONTACT: GARY THARALSON
(602) 438-3069

MOTOROLA ANNOUNCES A HIGH SPEED 64K STATIC RAM FAMILY

Austin, Texas, April 8, 1985... Motorola Memory Products Group announces a family of high-speed 64K Static Random Access Memories (RAMS), fabricted using Motorola's high performance second generation silicon-gate HCMOS III technology. The 8Kx8 Bit MCM6164, which will be sampled early third quarter 1985, will be followed by the MCM6188, with 16K X 4 Bit organization, and the b4K X 1 Bit MCM6187.

This 64K SRAM Family follows a high performance tradition recently established by the MCM6168, a 4K X 4 static RAM designed with 1.5 micron design rules to provide maximum density and reliability. These fully static RAMs contribute the speed necessary for cache memory, video applications, engineering work stations, and automated test equipment (ATE).

An improved address-transition-detection (ATD) technique is employed to optimize speed, achieving maximum access times of 70 nanoseconds for the MCM6164. The enhanced ATD design has been made impervious to address skew and fast voltage spikes.

The availability of postive- and negative-logic Chip Enable pins provides more system design flexibility than single Chip Enable devices. Output Enable increases data bus control. Operating from a single +5 volt (±10%) power supply, the fully static design eliminates the need for external clocks or timing strobes. Low maximum power consumptions inherent in HCMOS designs are maintained with 60 milliamps (mA) in active mode, 5mA maximum standby (TTL levels), and 2mA maximum standby at CMOS rail input levels.

These 64K fast static RAMs will be available in 600 mil, 28-pin plastic dual in-line packages (DIPs) with JEDEC standard pinout. Sampling for the MCM6164 is scheduled for July, 1985.

Reader Contact: Bob Crutchfield
512 928-7505

CERTIFIED SOFTWARE CORPORATION

616 CAMINO CABALLO, NIPOMO CA. 93444
USA TELEPHONE 805-349-0202 TELEX 467013

Larry Williams
'68' Micro Journal
5900 Cassandra Smith Rd.
Hixson, TN 37343

OMEGASOFT™ PASCAL

Thank you for your phone call the other day, I hope this short note will answer your questions.

First of all, Certified Software Corporation is the manufacturer of OmegaSoft Pascal and related products. We have recently moved to the address shown on the letterhead, please update your mailing list and subscription address.

We currently have five products to run on 6809 systems using the OS-9 (TM Microware), FLEX (TM TSC), and MDOS (TM Motorola) operating systems. These are the Pascal Compiler (including debugger), Relocating Assembler and Linking Loader, 9511 option, Screen Editor Kit, and Multi-Tasking Kernel. Our Pascal Compiler is based on the international standard but has many extensions for use in industrial control and other real-time applications.

We also have a PASCAL COMPILER package to run on 68000, 68008, and 68010 systems using the OS-9/68000 (TM Microware), VERSAdos (TM Motorola), and CP/M-68K (TM DRI) operating systems. This package includes the compiler, relocatable assembler and linking loader, screen editor, and several utilities. The debugger is currently under development and will be available by July 85.

I will contact you regarding a review of the 68000 product when the debugger is available. Please send current advertising rates and conditions to the above address.

Sincerely,

Robert Reimiller
President, Certified Software Corporation

FORTH Interest Group

FOR IMMEDIATE RELEASE
Linda Kahn--Publicist
(213) 478-7398

FREE FORTH INTEREST GROUP MEMBERSHIP AWARDED TO AUTHORS OF FORTH-RELATED ARTICLES

San Jose, CA, Apr. 17 -- Free FORTH Interest Group (FIG) membership is now available to authors of Forth-related articles. To qualify, just publish an article at least one page long in a non-Forth publication. Authors of Forth-related letters to an editor are eligible ffr a $10.00 membership discount.

The FORTH Interest Group is a worldwide non-profit member-supported organization with over 5,000 members and 70 chapters devoted to the Forth computer language. FIG membership of $20.00/year ($33.00 foreign) includes a subscription to FORTH Dimensions, a bi-monthly publication. FIG also provides an on-line data base, a job registry, a large selection of Forth literature, group health and life insurance, membership discounts, and many other services.

For additional information and a copy of the Author Recognition Guidelines call the FIG HOT LINE (415) 962-8653 or write FIG, P.O. Box 8231, San Jose, CA 95155.

NEW PRODUCT ANNOUNCEMENT

The MMI-A One Megabyte dynamic ram board for SS-50C bus systems is now available 1-2 weeks ARO. The MMI expands an SS-50 6809 system to its maximum available memory. On board switches allow disabling portions of memory for I/O and ROM space. Standard settings cover most applications. Special custom settings can be supplied for unusual situations. Price for the 2 Mhz version fully populated is $895.00, the 2.25 Mhz version is $995.00. Both versions run at their full rated speed with fully transparent refresh and are compatible with DMA disk controllers. Quantity discounts available. For more information contact: Dan Johnson (503) 244-8152.

JOHN ALFORD

I regret to inform that John Alford died March 19, 1985. Survived by his wife, Sally and their five year old daughter, Sarah.

John was best known for his 'SCREDITOR' series of editor/text processors, sold originally by ALFORD & ASSOC. and presently distributed by Windrush Micro Systems. Also John was one of the early designers of efficient digital speech systems and related microcomputer applications.

John was also known for his Christian remarks sections of his (Alford & Assoc.) advertising in various publications.

I first personally met John, Sally and daughter Sarah at one of the early Philly Computer Shows. And I can truthfully say that I never met a nicer person or family. John was sincere and honest in all his business dealings and I frequently received mail and other communications from various sources, remarking on his courtesy and attention to their needs. John Alford will be missed, not only by his family and friends, but by the entire 68XX community.

Professionally John Alford was one of the very best. But more important was the man - A good husband and father, an honest business man, an excellent example of his Christian faith and - a good friend. I will miss John Alford.

DMW

- - -

Classified Advertising

TELETYPE Model 43 PRINTER - with serial (RS232) interface, and full ASCII keyboard. LIKE NEW - New cost $1295.00 - ONLY $559.00 ready to run - Call Tom - Larry - Bob, CPI 615 842-4600

***

1 - SWTPC 6809 system in 6800 box - 56K, par, ser ports - $200
1 - SWTPC DMAI disk system - DMA 2 8" drives, case - $200
3 - CALDISK 8" drives - DS, DD - $50 ea
2 - SWTPC 6809 CPU board - $75 ea
1 - SWTPC 6809 CPU board - almost works - $25
1 - SWTPC AC30 cassette interface - $30
1 - JAMECO JE610 par keyboard, case - $40
1 - RADIO SHACK 26-2023 disk drive, case - $100
1 - DAIEL (IBM SELECTRIC I/O) printer,par - $150
2 - BASF 5" drives SS,DD for parts - $20 ea
4 - SWTPC (MOTOROLA) 32K dynamic memory - $75 ea
4 - SWTPC 8k static memory - $30 ea
3 - SWTPC MP-LA parallel I/O - $20 ea
1 - SWTPC MP-T timer - $20
1 - Stark CT-PS serial/parallel I/O - $20
1 - JPC IC-3 hi-speed cassette I/O - $20
1 - JPC CK-7 clock board - $20
1 - SWTPC MP-B3 mother board - $30
1 - F&D BMB-1 mother board - $30
1 - F&D MDC-1 I/O disk controller - $40
1 - Thomas 64x16 video board - $75
1 - Digital Research 32K static - $50
1 - OSD 32K static/EPROM - $50
1 - OSFT PASCAL - Color Computer - $40
all documentation - post paid on small items
Gil Shattuck Rt 2 Box 445 Hillsboro NH 03244 (603) 464-3850 evening

***

GIMIX 6809+ CPU with Gimix DAT, SSB DCB-4A disk controller, OS9 Level I with DCB-4A driver $995/offer
Dick Ollendorf, (201)852-6764 after 7PM

***

GIMIX pcbs, 8-parallel ports $80, 8-serial ports $120,touch-tone receiver $100, 16-button key path $35,6800-CPU with timers $60, votrax voice synt. $90.
Teletype-43 with RS-232-int. $250
Phone 201-662-1826

FEATURES THE POWERFUL, THIRD GENERATION, MOTOROLA 6809 PROCESSOR!

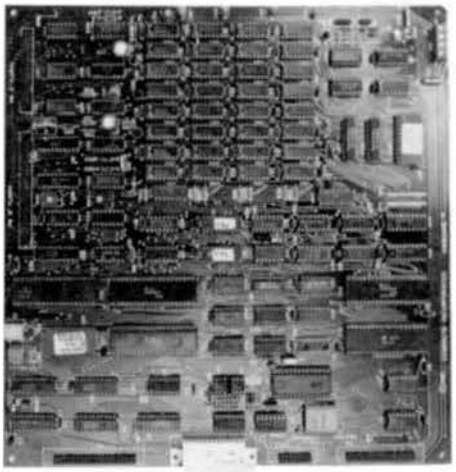
THE 6809 "UNIBOARD"™

SINGLE BOARD COMPUTER KIT

PERFECT FOR COLLEGES, OEM'S, INDUSTRIAL AND SCIENTIFIC USES!

64K RAM! DOUBLE DENSITY FLOPPY DISK CONTROLLER!

New! *Lower Price!*



BLANK PC BOARD

$99.95

WITH PAL'S, AND TWO EPROMS.

FOR 5-1/4 OR 8 INCH SOURCE DISKETTE ADD $10.

$219.00

COMPLETE KIT! FULLY SOCKETED.

PRICE CUT!!

THE COMPACTA UNIBOARD"™: Through special arrangement with COMPACTA INC., we are proud to have been selected the exclusive U.S. Mfg. of their new 6809 UNIBOARD"™ COMPUTER KIT. Many software professionals feel that the 6809 features probably the most powerful instruction set available today on ANY 8 bit micro. Now, at last, all of that immense computing power is available at a truly unbelievably low price.

ALL SALES ARE MADE SUBJECT TO THE TERMS OF OUR 90 DAY LIMITED WARRANTY. A FREE COPY IS AVAILABLE UPON REQUEST.

FEATURES:

- ★ 64K RAM using 4116 RAMS.
- ★ 6809E Motorola CPU.
- ★ Double Density Floppy Disk Controller for either 5-1/4 or 8 inch drives. Uses WD1793.
- ★ On board 80 x 24 video for a low cost console. Uses 2716 Char. Gen. Programmable Formats. Uses 6845 CRT Controller.
- ★ ASCII keyboard parallel input interface. (6522)
- ★ Serial I/O (6551) for RS232C or 20 MA loop.
- ★ Centronics compatible parallel printer interface. (6522)
- ★ Buss expansion interface with DMA channel. (6844)
- ★ Dual timer for real time clock application.
- ★ Powerful on board system monitor (2732). Features commands such as Go To, Alter, Fill, Move, Display, or Test Memory. Also Read and Write Sectors. Boot Normal, Unknown, and General Flex"™.

YOUR CHOICE OF POPULAR DISK OPERATING SYSTEMS:

FLEX"™ from TSC	$99
OS9"™ from Microware	$199

Specify 5-1/4 or 8 Inch

PC BOARD IS DOUBLE SIDED, PLATED THRU SOLDER MASKED, 11 x 11-1/2 IN.

Digital Research Computers
(OF TEXAS)
P.O. BOX 461565 · GARLAND, TEXAS 75046 · (214) 226-2309

TERMS: Shipments will be made approximately 3 to 6 weeks after we receive your order. VISA, MC, cash accepted. Add $4.00 shipping. USA AND CANADA ONLY

WINDRUSH MICRO SYSTEMS

UPROM II

PROGRAMS and VERIFIES: I2758, T2508, 12716, T2516, I2732/2732A, MCM68764/6, 12764/2764A, T2564, 127128/27128A, and I27256. I=Intel, T=Texas, M=Motorola.

NO PERSONALITY MODULES REQUIRED!

18E-VOLT EPROMS ARE NOt SUPPORTED

INTEL's Intelligent Programming (sm) implemented for Intel 2764, 27128 and 27256 devices. Intelligent programming reduces the average programming time of a 2764 from 7 minutes to 1 minute 15 seconds (under FLEX) with greatly improved reliability.

Fully enclosed pod with 5' of flat ribbon cable for connection to the host computer MC6821 PIA interface board.

MC6809 software for FLEX and OS9 (Level 1 or 2, Version 1.2).

BINARY DISK FILE offset loader supplied with FLEX, MDOS and OS9.

Menu driven software provides the following facilities:

a. FILL ..... a selected area of the buffer with a HEX char.
b. MOVE ..... blocks of data.
c. DUMP ..... the buffer in HEX and ASCII.
d. FIND ..... a string of bytes in the buffer.
e. EXAMINE/CHANGE . the contents of the buffer.
f. CRC ..... checksum a selected area of the buffer.
g. COPY ..... a selected area of an EPROM into the buffer.
h. VERIFY ..... a selected area of an EPROM against the buffer.
i. PROGRAM ..... a selected area of an EPROM with data in the buffer.
j. SELECT ..... a new EPROM type (return to types menu).
k. ENTER ..... the system monitor.
L RETURN ..... to the operating system.
L. EXECUTE ..... any DOS utility (only in FLEX and OS9 versions).

FLEX AND OS9 VERSIONS AVAILABLE FROM GIMIX. SS0/MDOS CONTACT US DIRECT.

MACE/XMACE/ASM05

ALL of these products feature a highly productive environment where the editor and the assembler reside in memory together. Gone are the days of tedius disk load and save operations while you are debugging your code.

- Friendly inter-active environment where you have instant access to the Editor and the Assembler, FLEX utilities and your system monitor.
- MACE can also produce ASMPROCs (GEN statements) for PL/9 with the assembly language source passed to the output as comments.
- XMACE is a cross assembler for the 6800/1/2/3/8 and supports the extended mnemonics of the 6303.
- ASM05 is a cross assembler for the 6805.

D-BUG

LOOKING for a single step tracer and mini in-line disassembler that is easy to use?? Look no further, you have found it. This package is ideal for those small assembly language program debugging sessions. D-BUG occupies less than 6K (including its stack and variables) and may be loaded anywhere in memory. All you do is LOAD IT, AIM IT and GO! (80 col VDUs only).

McCOSH 'C'

This is as complete a 'C' compiler as you will find on any operating system for the 6809. It is completely compatible with UNIX V7 and only lacks 'bit-fields' (which are of little practical use in an 8-bit world!).

- Produces very efficent assembly language source output with the 'C' source optionally interleaved as comments.
- Built-in optimizer will shorten object code by about 11%.
- Supports interleaved assembly language programs.
- INCLUDES its own assembler, the TSC relocating assembler is only required if you want to generate your own libraries.
- The pre-Processor, compiler, optimizer, assembler and loader all run independantly or under the 'CC' executive. 'CC' makes compiling a program to executable object as simple as typing in 'CC,HELLO.C <RETURN>'.

PL/9

- Friendly inter-active environment where you have INSTANT access to the Editor, the Compiler, and the Trace-Debugger, which, amongst other things, can single step the program a SOURCE line at a time. You also have direct access to any FLEX utility and your system monitor.
- 375+ page manual organized as a tutorial with plenty of examples.
- Fast SINGLE PASS compiler produces 8K of COMPACT and FAST 6809 machine code output per minute with no run-time overheads or license fees.
- Fully compatible with TSC text editor format disk files.
- Signed and unsigned BYTES and INTEGERS, 32-bit floating point REALs.
- Vectors (single dimension arrays) and pointers are supported.
- Mathematical expressions: (+), (-), (*), (/), modulus (\), negation (-)
- Expression evaluators: (=), (<>), (<), (>), (>=), (<=)
- Bit operators: (AND), (OR), (EOR/XOR), (NOT), (SHIFT), (SWAP)
- Logical operators: (.AND), (.OR), (.EOR/EOR)
- Control statements: IF..THEN..ELSE, IF..(CASE)..CASE2..ELSE, BEGIN..END, WHILE.., REPEAT..UNTIL, REPEAT..FOREVER, CALL, JUMP, RETURN, BREAK, GOTO.
- Direct access to (ACCA), (ACCB), (ACCD), (XREG), (CCR) and (STACK).
- FULLY supports the MC6809 RESET, NMI, FIRQ, IRQ, SWI, SWI2, and SWI3 vectors. Writing a self-starting (from power-up) program that uses ANY, or ALL, of the MC6809 interrupts is an absolute SNAP!
- Machine code may be embedded in the program via the 'GEN' statement. This enables you to code critical routines in assembly language and embed them in the PL/9 program (see 'MACE' for details).
- Procedures may be passed and may return variables. This makes them functions which behave as though they were an integral part of PL/9.
- Several fully documented library procedure modules are supplied: IOSUBS, BITIO, HARDIO, HEXIO, FLEXIO, SCIPACK, STRSUBS, BA TRING, and REALCOM.

'... THIS IS THE MOST EFFICIENT COMPILER I HAVE FOUND TO DATE.'

Quoted from Ron Andersons FLEX User Notes column in '68. Need we say more?

IEEE-488

- SUPPORTS ALL PRINCIPAL MODES OF THE IEEE-488 (1975/8) BUS SPECIFICATION:
  - talker
  - Listener
  - System Controller
  - Serial Poll
  - Parallel Poll
  - Group Trigger
  - Single or Dual Primary Address
  - Secondary Address
  - Talk only ... Listen only
- Fully documented with a complete reprint of the KILOBAUD article on the IEEE bus and the Motorola publication 'Getting aboard the IEEE Bus'.
- Low level assembly language drivers suitable for 6800, 6801, 6802, 6803, 6808 and 6809 are supplied in the form of listings. A complete back to back test program is also supplied in the form of a listing. These drivers have been extensively tested and are GUARANTEED to work.
- Single S-30 board (4, 8 or 16 addresses per port), fully socketed, gold plated bus connectors and IEEE interface cable assembly.

PRICES

D-BUG	(6809 FLEX only)	$ 75.00
MACE	(6809 FLEX only)	$ 75.00
XMACE	(6809 FLEX only)	$ 98.00
ASM05	(6809 FLEX only)	$ 98.00
PL/9	(6809 FLEX only)	$198.00
'C'	(6809 FLEX only)	$295.00
IEEE-488	with IEEE-488 cable assembly	$298.00
UPROM-II/U	with one version of software (no cable or interface)	$395.00
UPROM-II/C	as above but complete with cable and S-30 interface	$545.00
CABLE	5' twisted-flat 50 way cable with IDC connectors	$ 35.00
S-30 INT	SS-30 interface for UPROM-II	$130.00
EXOR INT	Motorola EXORbus (EXORciser) interface for UPROM-II	$195.00
UPROM SFT	Software drivers for 2nd operating system. Specify FLEX or OS9 AND disk size!	$ 35.00
UPROM SRC	Assembly language source (contact us direct)	

ALL PRICES INCLUDE AIR MAIL POSTAGE

Terms: CWO. Payment by Int'l Money Order, VISA or MASTER-CARD also accepted.

WORSTEAD LABORATORIES, NORTH WALSHAM, NORFOLK, ENGLAND. NR28 9SA.

TEL: 44 (692) 404086
TLX: 975548 WMICRO G

WE STOCK THE FOLLOWING COMPANIES PRODUCTS: GIMIX, SSB, FHL, MICROWARE, TSC, LUCIDATA, LLOYD I/O, & ALFORD & ASSOCIATES.

FLEX (tm) is a trademark of Technical Systems Consultants, OS-9 (tm) is a trademark of Microware Systems Corporation, MDOS (tm) and EXORciser (tm) are trademarks of Motorola Incorporated.

68' MICRO JOURNAL

Disk- 1 Filesort, Minicat, Minicopy, Minifms, **Lifetime, **Poetry, **Foodlist, **Diet.

Disk- 2 Diskedit w/ inst.& fixes, Prime, *Prmod, **Snoopy, **Football, **Hexpawn,**Lifetime

Disk- 3 Cbug09, Sec1, Sec2, Find, Table2, Intext, Disk-exp, *Disksave.

Disk- 4 Mailing Program, *Finddat, *Change, *Testdisk.

DISK- 5 *DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN, **BLACKJAK, **BOWLING.

Disk- 6 **Purchase Order, Index (Disk file indx)

Disk- 7 Linking Loader, Rload, Harkness

Disk- 8 Crtest, Lanpher (May 82)

Disk- 9 Datecopy, Diskfix9 (Aug 82)

Disk-10 Home Accounting (July 82)

Disk-11 Dissembler (June 84)

Disk-12 Modem68 (May 84)

Disk-13 *Initmf68, Testmf68, *Cleanup, *Dskalign, Help

Disk-14 *Init, *Test, *Terminal, *Find, *Diskedit, Init.Lib

Disk-15 Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Commo)

Disk-16 Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc, Date.Txt

Disk-17 Match Utility, RATBAS (A Basic Preprocessor) June 85

NOTE:

This is a reader service **ONLY!** No Warranty is offered or implied, they are **as received** by **'68' Micro Journal,** and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other.

PRICE: 8" Disk **$14.95** - 5" Disk **$12.95**

68' Micro Journal
5900 Cassandra Smith Rd.
Hixson, Tn. 37343
(615)842-4600

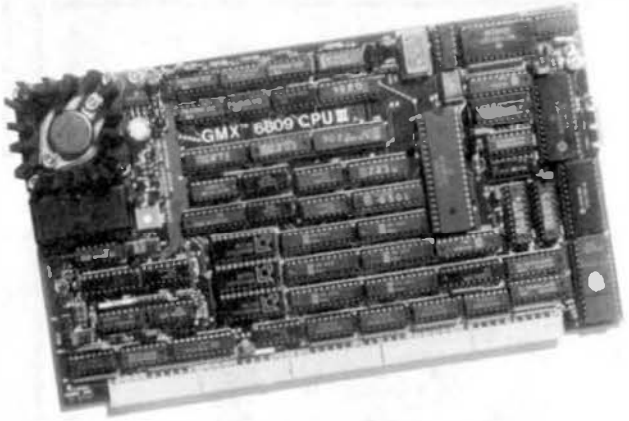
* Indicates 6800
** Indicates BASIC SWTPC or TSC
6809 no Indicator.

MASTER CARD - VISA Accepted
Foreign -- add **10%** for Surface
or **20%** for Air!!

68' MICRO JOURNAL ADVERTISERS INDEX

This index is provided as a reader service. The publisher does not assume any liability for omissions or errors.

GIMIX STATE OF THE ART 6809 SYSTEMS FOR THE SERIOUS USER.



GIMIX has 19MB or high performance 47MB Winchester Drive Systems and/or Floppy Disk Drive Systems.

For the user who appreciates the need for a bus structured system using STATIC RAM and powered by a ferro resonant constant voltage transformer.

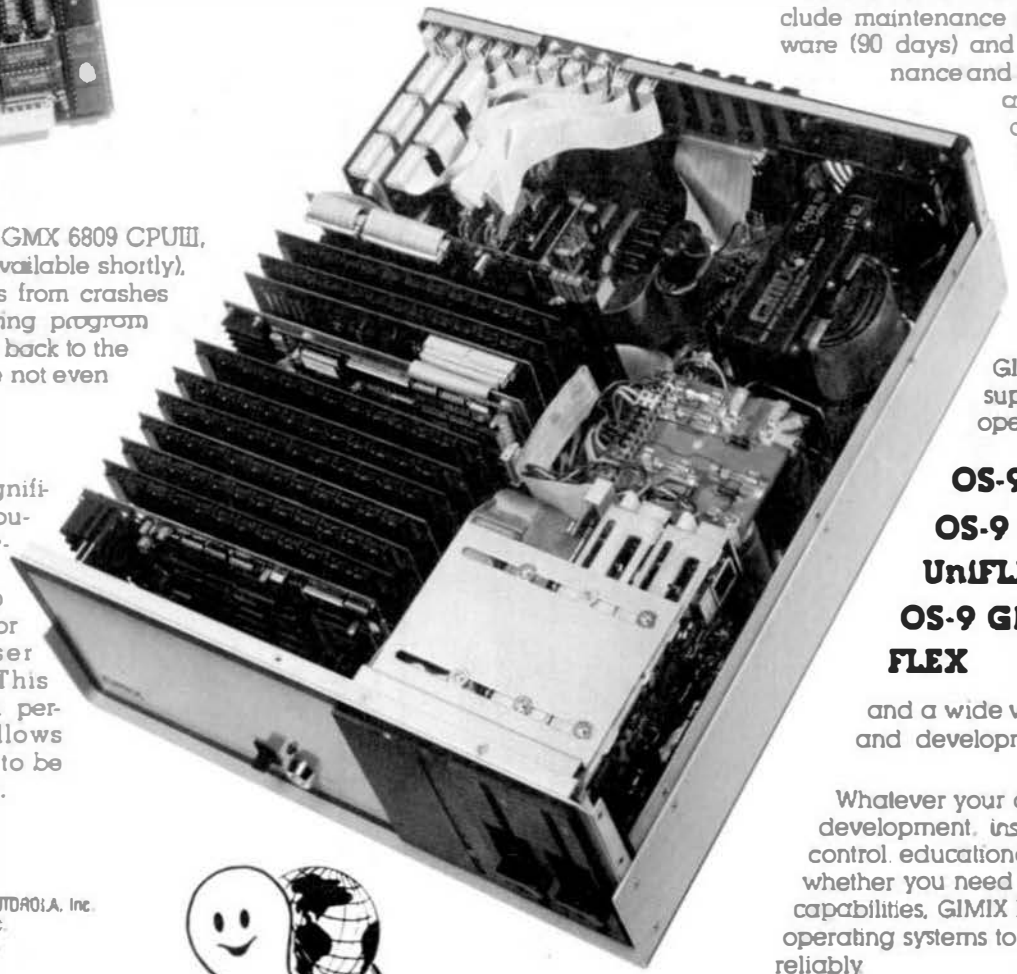
GIMIX has single user systems that can run both FLEX and OS-9 or Multi user systems for use with UniFLEX or OS-9.

GIMIX versions of OS9 and UniFLEX include maintenance and support by Microware (90 days) and TSC (1 year). Maintenance and support after this period are available at extra cost.

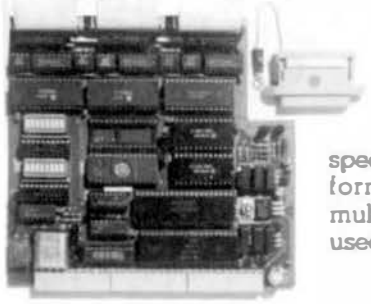
(NOTE: this support and maintenance is only for use with approved GIMIX hardware)

'68' Micro Journal

For the ultimate in performance, the Unique GMX 6809 CPUIII, using either OS-9 GMXIII or UniFLEX GMXIII (available shortly), gives protection to the system and other users from crashes caused by defective user programs. e.g. During program development, a programmer who crashes goes back to the shell or the debugger, while the other users are not even aware anything occurred.



The intelligent serial I/O processor boards significantly reduce system overhead by handling routine I/O functions, thereby freeing up the host CPU for running user programs. This speeds up system performance and allows multiple terminals to be used at 19.2K baud.



GIMIX 6809 systems support five predominant operating systems:

OS-9 GMX III,
OS-9 GMX II,
UniFLEX,
OS-9 GMX I,
FLEX

and a wide variety of languages and development software.

Whatever your application: software development, instrumentation, process control, educational, scientific or business, whether you need single or multi-user capabilities, GIMIX has hardware and the operating systems to get the job done reliably.

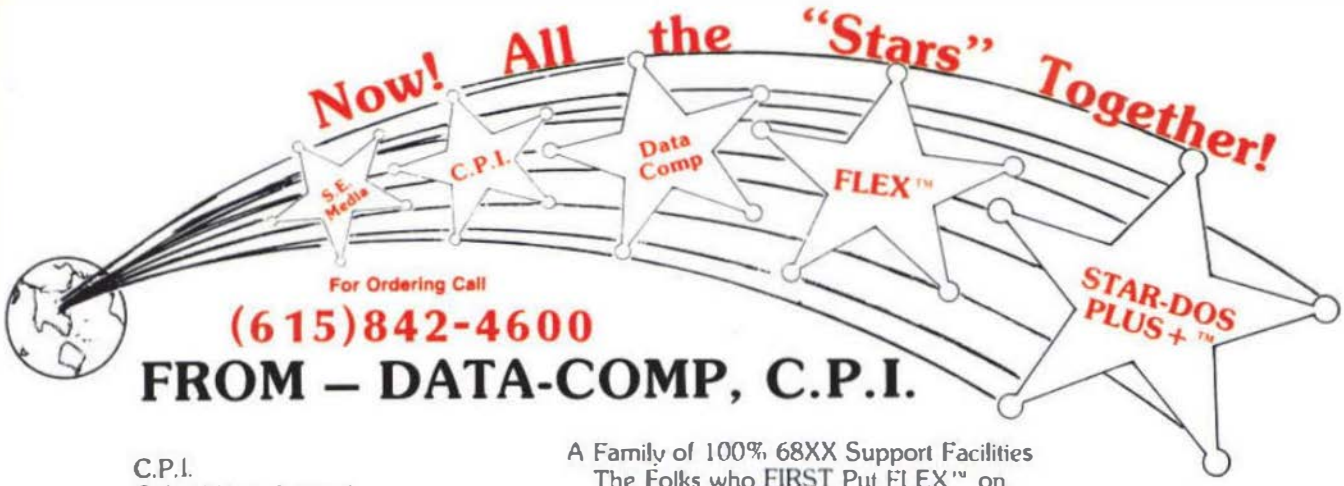
Please phone or write if you need further information.

July '85

GIMIX INC.

1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX 910-221-4055

64



For Ordering Call

(615)842-4600

FROM – DATA-COMP, C.P.I.

C.P.I.
Color Micro Journal
'68' Micro Journal
Data-Comp
S.E. Media

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo
Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00
Only $79.95

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.50

FLEX-CoCo Jr.
without TSC
Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ Many Many More!!!
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

NEW LOWER PRICES ON PAK #5, AND PRINTERS

THESE PACKAGES INCLUDE DRIVE, *CONTROLLER, POWER SUPPLY & CABINET, CABLE, AND MANUAL.

* SPECIFY WHAT CONTROLLER YOU WANT J&M, OR RADIO SHACK.

PAK #1 - 1 SINGLE SIDED, DOUBLE DENSITY SYS.	$349.95
PAK #2 - 2 SINGLE SIDED, DOUBLE DENSITY SYS.	$639.95
PAK #3 - 1 DOUBLE SIDED, DOUBLE DENSITY SYS.	$439.95
PAK #4 - 2 DOUBLE SIDED, DOUBLE DENSITY SYS.	$699.95
PAK #5 - 2 DOUBLE SIDED, DOUBLE DENSITY SYS. THINLINE DRIVES, HALF SIZE	$499.95

Controllers

J&M DISK CONTROLLER W/ DOS OR RADIO SHACK DISK BASIC, SPECIFY WHAT DISK BASIC.	$134.95
RADIO SHACK DISK CONTROLLER 1.1	$134.95

Misc.

64K UPGRADE W/MOD. INSTRUCTIONS, C,D,E,F, AND COCO 2	$ 44.95
HJL KEYBOARDS	$ 74.95
MICRO TECH LOWER CASE ROM ADAPTER	$ 74.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET & POWER SUPPLY	$ 49.95
SINGLE SIDED, DOUBLE DENSITY 5" DISK DRIVE	$199.95
DOUBLE SIDED, DOUBLE DENSITY 5" DISK DRIVE	$249.95

Printers

EPSON RX-80	$269.00
EPSON RX-80FT	$369.00
EPSON MX-100	$499.00
EPSON FX-100	$799.00
EPSON FX-80	$549.00
EPSON MX-70	$200.00

Disk Drive Cables

CABLE FOR ONE DRIVE	$ 19.95
CABLE FOR TWO DRIVES	$ 24.95

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
TELEX 558 414 PVT BTH

68 Micro Journal
ISSN 0194-5025
5900 Cassandra Smith Rd.
Hixson, TN 37343

Second Class Postage Paid
At Hixson, TN
And Additional Mailing Offices